063 — 557 — 87 GAJ
03 Kčs 16,50

GRAMATIKY A JAZYKY

alfa
SNTL

Ľ. Molnár
M. Češka
B. Melichar

GRAMATIKY A JAZYKY

alfa
SNTL

GRAMATIKY A JAZYKY

Jedným z kľúčových pojmov vo výpočtovej technike je pojem programovací jazyk. Spôsob práce počítačov vyžaduje, aby programovací jazyk bol vhodne definovaný a špecifikovaný. Formalizmus definovania a špecifikácie programovacích jazykov vychádza z teórie, ktorú inicializoval v roku 1956 Noam Chomsky vytvorením matematického modelu gramatiky. Teória formálnych jazykov patrí v súčasnosti k najprepracovanejším oblastiam výpočtovej techniky.

Vysokoškolská učebnica je venovaná špecifikačným prostriedkom jazykov, gramatikám a automatom. Po zavedení Chomského hierarchie jazykov sa zameriava na regulárne a bezkontextové jazyky, ktoré majú najširšie použitie v oblasti výpočtovej techniky, a to najmä pri návrhu a špecifikácii programovacích jazykov a prekladačov. Preto sa veľká pozornosť venuje deterministickým jazykom, najmä jazykom LL(k) a LR(k), ich syntaktickej analýze a automatickej konštrukcii syntaktických analyzátorov.

Vychádza z osnov rovnomenného predmetu Gramatiky a jazyky študijného odboru elektronické počítače vysokých škôl technických. Preto je určená predovšetkým študentom tohto odboru. Obsahovo však vyhovuje i študentom iných vysokých škôl zameraných na výpočtovú techniku, ako aj ostatným záujemcom.

GRAMATIKY A JAZYKY

alfa
SNTL

GRAMATIKY A JAZYKY

GRAMATIKY
A JAZYKY

ĽUDOVÍT MOLNÁR
MILAN ČEŠKA
BOŘIVOJ MELICHAR

GRAMATIKY A JAZYKY

ALFA — VYDAVATEĽSTVO TECHNICKEJ A EKONOMICKEJ LITERATÚRY BRATISLAVA

SNTL — NAKLADATELSTVÍ TECHNICKÉ LITERATURY PRAHA

Využívanie výpočtovej techniky je podmienené kvalitným programovým vybavením, v ktorom je kľúčový pojem programovací jazyk. Formalizmus definovania a špecifikácie programovacích jazykov vychádza z teórie vytvorenia matematického modelu gramatiky. Hoci teória formálnych jazykov patrí v súčasnosti k najprepracovanejším oblastiam výpočtovej techniky, nie je ešte literatúrou dostatočne pokrytá.

Vysokoškolská učebnica vychádza z osnov rovnomenného predmetu Gramatiky a jazyky študijného odboru elektronické počítače vysokých škôl technických. Preto je určená predovšetkým študentom tohto odboru. Obsahovo však vyhovuje aj študentom ostatných vysokých škôl zameraných na výpočtovú techniku a pracovníkom z praxe.

Schválilo Ministerstvo školstva SSR po dohode s Ministerstvom školstva ČSR dňa 8. 7. 1986 číslo Š 6981/1986-30 ako celoštátnu vysokoškolskú učebnicu pre elektrotechnické fakulty vysokých škôl.

Lektorovali: Ing. Karel Ježek, CSc.
Doc. RNDr. Branislav Rovan, CSc.

Redakcia matematiky, fyziky a výpočtovej techniky — vedúca redaktorka Anna Známová

OBSAH

PREDHOVOR

Riešenie jazykových problémov komunikácie človeka s počítačom podnietilo záujem jazykovedcov o vybudovanie matematickej teórie jazykov. Výskum v tejto oblasti skutočne vyústil do ucelenej teórie tzv. formálnych jazykov, ktorej základom je matematický model gramatiky. Vytvorenie matematického modelu gramatiky nielenže umožnilo hlbšie štúdium jazykových problémov, ale malo bezprostredný účinok na riešenie praktických problémov spojených s definovaním i prekladom programovacích jazykov. Predovšetkým aplikovateľnosť teórie formálnych jazykov a ich priamy súvis s teóriou automatov, ako druhým abstraktným systémom na opis jazykov, spôsobili, že sa táto problematika dostala aj do učebných osnov študijných odborov elektronické počítače na vysokých školách technických.

Vysokoškolská učebnica, ktorá sa vám dostáva do rúk, vychádza z osnov rovnomenného predmetu Gramatiky a jazyky. Jej napísaním poverili Ministerstvá školstva SSR a ČSR autorský kolektív v zložení doc. RNDr. ĽUDOVÍT MOLNÁR, CSc., doc. RNDr. MILAN ČEŠKA, CSc., a doc. Ing. BOŘIVOJ MELICHAR, CSc. Uvedený autorský kolektív sa spoločne podielal na vytvorení koncepcie, logickej i obsahovej stavbe predkladanej učebnice, preto ju v tomto smere treba chápať ako spoločné dielo. Podrobné rozpracovanie jednotlivých častí si autori rozdelili takto: kapitoly 1, 2 a 3 a článok 4.1 napísal Ľ. MOLNÁR, zvyšok kapitoly 4 a kapitolu 5 M. ČEŠKA, kapitolu 6 B. MELICHAR. Za koordináciu celej učebnice bol zodpovedný Ľ. MOLNÁR.

Obsahovú stránku učebnice sme sa snažili koncipovať tak, aby študenti získali základné vedomosti z teórie formálnych jazykov a teórie automatov a aby ich vedeli prakticky použiť pri definovaní a analýze predovšetkým programovacích jazykov. Či je takáto koncepcia i spôsob jej spracovania správny, ukážu až praktické skúsenosti z jej používania. Budeme preto vďační za každú podnetnú pripomienku, ktorú by sme mohli zohľadniť pri prípadnom ďalšom vydaní učebnice alebo priamo v pedagogickom procese.

Záverom by sme sa chceli poďakovať lektorom doc. RNDr. B. ROVANOVI, CSc., a Ing. K. JEŽKOVI, CSc., za starostlivé prečítanie rukopisu a celý rad konštruktívnych návrhov, ktoré skvalitnili jeho pôvodnú verziu. Naša vďaka patrí aj pracovníkom vydavateľstva Alfa, predovšetkým RNDr. J. BELASOVEJ, za porozumenie a pomoc pri príprave vydania tejto učebnice.

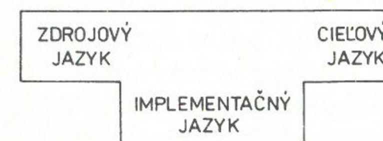
Autori

1 ÚVOD

1.1 PREKLADAČ A JEHO ŠTRUKTÚRA

Aj keď formálne jazyky a systémy pre ich špecifikáciu možno skúmať z rôznych hľadísk, nás budú zaujímať v prvom rade z hľadiska ich použitia a využitia pri špecifikácii prekladačov. Pri programovacích jazykoch nás zaujímajú tak ich vyjadrovacie schopnosti, ako aj vlastnosti z hľadiska ich rozpoznania. Prvá vlastnosť sa týka programovania, druhá prekladu. Jazyk dnes nemožno navrhovať bez toho, aby sme nebrali do úvahy obidve tieto vlastnosti. V tomto článku sa nebudeme zaoberať návrhom jazykov, ale skôr ich prekladom. Z tohto pohľadu si hneď v úvode povieme, čo je funkciou prekladača a aká je jeho štruktúra. To nám môže slúžiť ako motivácia pre skúmanie viacerých stránok jazykov.

Prevažná väčšina programov sa v súčasnosti programuje vo vyšších programovacích jazykoch. Takéto programovanie je bližšie používateľovi, ale má i celý rad ďalších výhod, ktorými sa tu nebudeme zaoberať. Realizácia programov vo vyššom programovacom jazyku si však vyžaduje, aby takýto program bol preložený z daného vyššieho jazyky do „materinského" jazyka toho-ktorého počítača. Túto funkciu zabezpečuje *prekladač*. Prekladač je program, ktorý číta program vo vyššom programovacom jazyku — *zdrojový program*, a produkuje *cieľový program*. Hovoríme aj o *zdrojovom* a *cieľovom jazyku*. Keďže prekladač je tiež program, musí byť zapísaný v určitom jazyku. Tento jazyk nazývame *implementačný*. Z pohľadu týchto troch jazykov sa prekladač zvykne zobrazovať v tvare písmena T. (*obr. 1.1*).



Obr. 1.1. Prekladač z hľadiska použitých jazykov

Proces prekladu sa uskutočňuje konštruktívnym spôsobom. Z jazykového hľadiska ho možno rozdeliť na tieto hlavné časti:

a) lexikálna analýza,
b) syntaktická analýza,
c) spracovanie sémantiky,
d) generovanie cieľového programu.

Aby sa preklad mohol realizovať, musí byť program vytvorený podľa určitých pravidiel vyplývajúcich z definície jazyka. To je teda prvé uplatnenie teórie formálnych jazykov. Úlohou analýzy programu je práve zistiť, či je program vytvorený podľa týchto pravidiel. Navyše, získané informácie určitým spôsobom využívame na ďalšiu činnosť prekladu.

Analýza je rozdelená na dve hierarchické úrovne. V lexikálnej analýze zabezpečujeme jednak vstup programu a z jeho jednotlivých znakov vytvárame vyššie jednotky — symboly, ako je napr. číslo, identifikátor, obmedzovač a pod. Každý symbol musí byť vytvorený podľa pravidiel daného jazyka, a to tak z hľadiska prípustnosti znakov, ako aj z hľadiska štruktúry symbolu. Ako vieme, zo symbolov jazyka sú vytvorené vyššie jednotky jazyka, ako je napr. podmienený príkaz, príkaz priradenia, cyklus a pod. Úlohou syntaktického analyzátora je kontrola správnosti týchto vyšších jednotiek s uchovaním niektorých získaných informácií o štruktúre skúmanej syntaktickej jednotky.

Po syntaktickej analýze sa vykonáva ešte sémantická analýza a urobí sa predpríprava pre tvorbu finálneho produktu — prekladu, ktorá spočíva vo vhodnej transformácii pôvodného programu na určitú vnútornú formu.

Posledná časť prekladu zabezpečuje generovanie cieľového programu.

Rozdelenie analýzy na lexikálnu a syntaktickú má svoje dôvody práve v teórii formálnych jazykov. Keďže jazyk symbolov je podstatne jednoduchší, ide o regulárny jazyk, možno lexikálnu analýzu realizovať jednoduchšími a účinnejšími prostriedkami. Navyše, na základe prostriedkov pre špecifikáciu regulárnych jazykov, regulárnych gramatík a konečných automatov možno skonštruovať aj vlastný lexikálny analyzátor.

Vyššie jednotky programovacích jazykov majú v porovnaní so štruktúrou symbolov zložitejšiu štruktúru, a preto si vyžadujú aj zložitejšie špecifikačné prostriedky. Špecifikujeme ich pomocou bezkontextových gramatík a zásobníkových automatov. Aj keď ide o podstatne zložitejší aparát, možno ho využiť na konštrukciu syntaktických analyzátorov.

Využitie formalizmu gramatík a automatov na konštrukciu analyzátorov je veľmi dôležité preto, lebo túto činnosť možno robiť mechanicky, a teda ju prenechať počítaču. Takáto automatická konštrukcia vybraných častí prekladačov nielen skráti jeho tvorbu, ale zvýši aj jeho spoľahlivosť.

V ďalších článkoch sa budeme podrobnejšie venovať niektorým otázkam formálnych jazykov a systémov na ich špecifikáciu.

1.2 ABECEDA A FORMÁLNY JAZYK

Už pri vyučovaní prirodzeného jazyka začíname prvotnými pojmami hláska, písmeno, z ktorých vytvárame vyššie jednotky — slová a vety. Podobná situácia je aj pri formálnych jazykoch. Aj tu vychádzame z prvotných pojmov, ktorými sú prípustné symboly tvoriace abecedu jazyka. Zo symbolov vytvárame vyššie jednotky — reťazce alebo vety — ako vhodné postupnosti prípustných symbolov.

Abeceda je konečná množina prvkov, ktoré nazývame symboly.

Príklad 1.1.

{A, B, C, ..., Z}	latinka
{0, 1, 2, 3, ..., 9}	abeceda desiatkových číslic
{I, V, X, L, C, D, M}	abeceda rímskych číslic
{+, −, ×, /, (,), =, ␣}	abeceda, ktorá spolu s latinkou a abecedou desiatkových číslic tvorí abecedu jazyka fortran

{**and**, **array**, **begin**, **case**, **const**, **div**, **do**, **downto**, **else**, **end**, **file**, **for**, **function**, **goto**, **if**, **in**, **label**, **mod**, **nil**, **not**, **of**, **or**, **packed**, **procedure**, **program**, **record**, **repeat**, **set**, **then**, **to**, **type**, **until**, **var**, **while**, **with**} je množina vyhradených slov jazyka pascal.

Ľubovoľnú konečnú postupnosť pozostávajúcu zo symbolov danej abecedy nazývame *reťazec*. Počet symbolov v reťazci nazývame *dĺžka reťazca*, dĺžku reťazca x budeme označovať $|x|$. Pochopiteľne, že význam majú iba reťazce konečnej dĺžky. Preto v ďalšom texte budeme pod reťazcom rozumieť vždy reťazec konečnej dĺžky. Reťazec s nulovou dĺžkou nazývame *prázdny reťazec* a budeme ho označovať e. Ak $x = a_1a_2 \ldots a_n$ je reťazec a a_i sú symboly, tak *obráteným reťazcom* nazývame reťazec $x^R = a_na_{n-1} \ldots a_1$.

Príklad 1.2. Nech $A = \{a, b, c, d\}$ je nejaká abeceda. Nad danou abecedou potom možno vytvoriť tieto reťazce:

$x_1 = aab$	s dĺžkou $\|x_1\| = 3$
$x_2 = a$	s dĺžkou $\|x_2\| = 1$
$x_3 = abcd$	s dĺžkou $\|x_3\| = 4$
$x_4 = e$	s dĺžkou $\|x_4\| = 0$
$x_5 = baa$	s dĺžkou $\|x_5\| = 3$
$x_1^R = baa$	
$x_2^R = a$	
$x_1 = x_5^R$	

Na množine reťazcov danej abecedy možno definovať operáciu zreťazenia s operátorom . , a to nasledujúcim spôsobom: Ak $u = a_1a_2 \ldots a_k$, $v = b_1b_2 \ldots b_l$ sú reťazce nad danou abecedou, tak $u \,.\, v = a_1a_2 \ldots a_kb_1b_2 \ldots b_l$. Zreťazením reťazcov u, v dostaneme nový reťazec uv. Zápisy uv a $u \,.\, v$ sú teda ekvivalentné.

Ľahko možno zistiť, že reťazce vytvorené zreťazením sú opäť reťazce nad danou abecedou. Množina všetkých reťazcov nad danou abecedou je teda vzhľadom na operáciu zreťazenie uzavretá. Ak x, y, z sú ľubovoľné reťazce nad určitou abecedou, tak x nazývame *prefix* alebo *predpona* a y *postfix* alebo *prípona* reťazca xy. Reťazec y je *podreťazcom* reťazca xyz pre nejaké reťazce x, z. Vidíme, že prefix aj postfix sú podreťazcami daného reťazca. Ak A je nejaká abeceda, tak množinu všetkých reťazcov nad danou abecedou vrátane prázdneho reťazca budeme označovať A^*. Množinu všetkých reťazcov bez prázdneho reťazca A^+, teda $A^+ = A^* - \{e\}$.

Príklad 1.3. Nech $A = \{0, 1\}$ je abeceda. Potom možno vytvoriť reťazce:

$x_1 = 0101$
$x_2 = 111$
$x_3 = 110$
$x_1 . x_2 = 0101111$
$x_2 . x_3 = 111110$

Reťazec $x = 101$ môže mať nasledujúce prefixy, postfixy a podreťazce:

prefix: e, 1, 10, 101
postfix: e, 1, 01, 101
podreťazec: e, 1, 10, 101, 01, 0

$A^* = \{e, 0, 1, 00, 01, 10, 11, 000, \ldots\}$
$A^+ = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$

Z uvedeného príkladu vidíme, že prázdny reťazec e je prefix, postfix aj podreťazec ľubovoľného reťazca. Podreťazcom reťazca 101 však nie je napr. reťazec 11, pretože neexistujú žiadne také reťazce x, y, aby sa $x11y$ rovnalo 101.

Definícia 1.1. Ak je daná abeceda A, tak ľubovoľnú podmnožinu množiny A^* nazývame *formálny jazyk nad abecedou A*.

Uvedená definícia jazyka je príliš všeobecná. Vzniká prirodzená otázka, ako jazyk opísať a ako ho špecifikovať. Zo špecifikácie množín vieme, že ich možno špecifikovať buď vymenovaním všetkých prvkov, alebo pomocou definovania vlastnosti, ktorú každý prvok musí mať. Keďže jazyk je množinou prvkov — reťazcov, možno ho tiež špecifikovať jedným z uvedených spôsobov.

Ak je množina reťazcov daného jazyka konečná, nazývame ho *konečný jazyk*, ak je táto množina prázdna, nazývame ho *prázdny jazyk*, ak je táto množina nekonečná, jazyk nazývame *nekonečný*.

Príklad 1.4. Nech $A = \{a\}$ je abeceda. Jazyk pozostávajúci zo všetkých reťazcov nad abecedou A vrátane prázdneho reťazca možno potom špecifikovať takto:

$L_1 = \{a^i / i \geq 0\}$

kde $a^n = \underbrace{aa \ldots a}_{n\text{-krát}}$,

Možno však využiť aj náš zápis pre množinu všetkých reťazcov nad danou abecedou a špecifikovať jazyk L_1 takto:

$L_1 = \{a\}^*$

pričom sa často používa skrátený zápis a^*. Jazyk L_1 je nekonečný. Podobne možno špecifikovať jazyk, ktorý obsahuje iba prázdny reťazec, ako

$L_0 = \{e\}$

prázdny jazyk $L_\emptyset = \emptyset$.

Keďže abecedy i jazyky sme definovali ako množiny, platia pre ne bežné množinové operácie. Ak A, B sú abecedy a L_1, L_2 jazyky, pričom $L_1 \subset A^*$, $L_2 \subset B^*$, platí

$A \cup B = \{x/x \in A \vee x \in B\}$ $\quad L_1 \cup L_2 = \{y/y \in L_1 \vee y \in L_2\}$
$A \cap B = \{x/x \in A \wedge x \in B\}$ $\quad L_1 \cap L_2 = \{y/y \in L_1 \wedge y \in L_2\}$
$A - B = \{x/x \in A \wedge x \notin B\}$ $\quad L_1 - L_2 = \{y/y \in L_1 \wedge y \notin L_2\}$

Okrem uvedených operácií sa pri jazykoch často stretávame s operáciami *zreťazenie* a *iterácia*.

Definícia 1.2. Ak L_1 a L_2 sú jazyky, ich zreťazením dostaneme jazyk L, pričom

$L = \{xy/x \in L_1 \wedge y \in L_2\}$

Príklad 1.5. Nech $L_1 = \{A, B, \ldots, Z\}$ a $L_2 = \{A, B, \ldots, Z, 0, 1, \ldots, 9\}^*$. Potom jazykom $L = L_1L_2$ je množina všetkých reťazcov nad abecedou $\{A, B, \ldots, \ldots, Z, 0, 1, \ldots, 9\}$, ktoré začínajú písmenom. Jazykom L je teda množina bežne používaných indentifikátorov. Na základe operácie zreťazenia je definovaná iterácia jazyka.

Definícia 1.3. Ak L je jazyk, tak jeho n-tú mocninu definujeme takto:

$L^0 = \{e\}$
$L^n = LL^{n-1}$ pre $n \geq 1$

Iteráciou jazyka L nazývame množinu L^*, kde

$$L^* = \bigcup_{n \geq 0} L^n$$

Pozitívnou iteráciou jazyka L nazývame množinu L^+, kde

$$L^+ = \bigcup_{n \geq 1} L^n$$

Ľahko zistíme, že

$$L^+ = LL^* = L^*L$$
$$L^* = L^+ \cup \{e\}$$

Príkad 1.6. Nech $L = \{0, 1\}$. Potom

$$L^0 = \{e\}$$
$$L^1 = \{0, 1\}$$
$$L^2 = \{00, 01, 10, 11\}$$
$$L^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

Okrem uvedených operácií nás budú zaujímať aj zobrazenia definované nad jazykmi.

Definícia 1.4. Nech A_1, A_2 sú abecedy. Potom ľubovoľné zobrazenie h: $A_1 \to A_2^*$ budeme nazývať *homomorfizmus*. Možno ho rozšíriť z množiny symbolov A_1 na množinu reťazcov A_1^*, a to takto: $h(e) = e$, $h(ua) = h(u)h(a)$ pre všetky $u \in A_1^*$, $a \in A_1$.

Aplikáciou homomorfizmu h na jazyk L dostaneme nový jazyk $h(L)$, kde $h(L) = \{h(w)/w \in L\}$.

Príklad 1.7. Majme jazyk $L = \{0, 1, \ldots, 9\}^*$. Ide o jazyk celých nezáporných čísel bez znamienka s vedúcimi nulami. Zobrazením h definovaným ako $h(0) = 0000$, $h(1) = 0001, \ldots, h(8) = 1000$, $h(9) = 1001$, použitým na L, dostaneme jazyk $h(L)$, v ktorom každý reťazec je dvojkovo desiatkovým kódom reťazca z L. Teda napr. reťazec $904 \in L$ a $h(904) = 100100000100 \in h(L)$.

Definícia 1.5. Ak h: $A_1 \to A_2^*$ je homomorfizmus, tak h^{-1}: $A_2^* \to 2^{A_1^*}$ je *inverzný homomorfizmus*, ktorý je definovaný takto: Ak $v \in A_2^*$, tak $h^{-1}(v)$ je množina reťazcov nad A_1, ktoré sa homomorfizmom h zobrazujú na v, teda $h^{-1}(v) = \{u/h(u) = v\}$. Ak L je jazyk nad A_2, tak $h^{-1}(L)$ je jazyk nad A_1 pozostávajúci z reťazcov, ktoré h zobrazuje na reťazce z L.

$$h^{-1}(L) = \bigcup_{v \in L} h^{-1}(v) = \{u/h(u) \in L\}$$

Aby špecifikácia jazyka mala praktický význam, musí byť konečná. V súčasnosti sa najčastejšie používajú dva systémy, ktoré spĺňajú požiadavku konečnosti. Prvý systém, automaty, je založený na princípe „procedúry", ktorá po predložení reťazca po konečnom počte krokov rozpozná, či tento reťazec patrí alebo nepatrí do daného jazyka. Druhý systém, gramatiky, je schopný generovať iba také reťazce, ktoré patria do daného jazyka a je schopný generovať všetky reťazce daného jazyka. Uvedenými systémami sa budeme v ďalšom zaoberať podrobnejšie.

Cvičenia

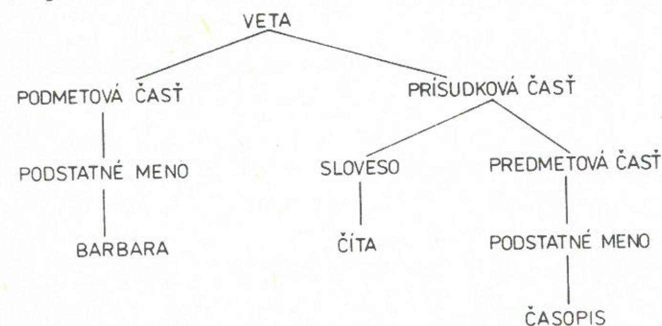
1. Napíšte množiny štandardných konštánt, typov, súborov a funkcií jazyka pascal a pre každý prvok ako reťazec
   a) určte jeho dĺžku,
   b) vytvorte obrátený reťazec,
   c) určte všetky prefixy a postfixy,
   d) zistite, či je podreťazcom iného reťazca danej množiny.
2. Napíšte množinu všetkých binárnych reťazcov, ktoré možno zobraziť v jednom bajte.
3. Napíšte množinu všetkých reťazcov nad abecedou $\{0, 1\}$ dĺžky osem.
4. Nech $L = \{0, 1\}$. Napíšte množinu L^8.

2 FORMÁLNE JAZYKY A GRAMATIKY

V tejto kapitole sa budeme zaoberať generatívnym spôsobom špecifikácie jazyka, ktorý je reprezentovaný formálnym systémom — gramatikou. Prv než pristúpime k jej definícii a k zisťovaniu jej vlastností, rozoberieme problém špecifikácie jazyka generatívnym spôsobom na prirodzenom jazyku. Aj keď v prirodzenom jazyku možno za prvotný symbol považovať písmeno, z ktorého potom vytvárame slová a vety, prirodzenejšie je považovať za prvotný symbol slovo a z neho vytvárať vety. Ako abeceda tu teda bude množina slov. Nebude nás zaujímať štruktúra slov, ale štruktúra vety. Ešte raz zdôrazňujeme terminológiu: symbolu zodpovedá slovo, reťazcu veta.

Štruktúra vety prirodzeného jazyka je definovaná gramatickými pravidlami. Bežnú slovenskú vetu možno gramaticky definovať ako *podmetovú časť*, po ktorej nasleduje *prísudková časť*. Vetu teda definujeme pomocou ďalších dvoch jednotiek, ktorých význam zatiaľ nepoznáme, a preto ich musíme definovať. Podmetovú časť možno definovať ako *podstatné meno* a prísudkovú časť ako *sloveso*, za ktorým nasleduje *predmetová časť*.

Základnou vlastnosťou doterajšej analýzy je, že niektoré jednotky jazyka definujeme pomocou iných jednotiek. Aby bol jazyk správne definovaný, musí tento proces byť taký, aby sme v konečnom dôsledku definovali každú jednotku



Obr. 2.1. Gramatická štruktúra vety prirodzeného jazyka

pomocou veličín, ktoré sú známe, prvotné, ktoré sa nakoniec objavia v skutočnej vete. Tak by sme mohli podstatné meno nahradiť skutočným podstatným menom, sloveso konkrétnym slovesom atď. Celý proces možno pre vetu „Barbara číta časopis" graficky zobraziť na *obr. 2.1.*

Ako vidieť z obrázka, prvotné jednotky sú na najnižšej úrovni. Na odlíšenie prvotných jednotiek od jednotiek definovaných pomocou iných jednotiek používame zvyčajne špeciálne ohraničujúce symboly, odlišné typy písma a pod. Jedným z bežných spôsobov je uzatváranie definovaných jednotiek do lomených zatvoriek ⟨, ⟩.

Gramatické pravidlá, pomocou ktorých definujeme jednotlivé jednotky, budeme písať v tomto tvare:

⟨jednotka, ktorú definujeme⟩ → jednotky, pomocou ktorých definujeme jednotku na ľavej strane

Vetu prirodzeného jazyka možno teraz definovať nasledujúcimi gramatickými pravidlami:

1. ⟨veta⟩ → ⟨podmetová časť⟩ ⟨prísudkova časť⟩
2. ⟨podmetová časť⟩ → ⟨podstatné meno⟩
3. ⟨podstatné meno⟩ → BARBARA
4. ⟨prísudková časť⟩ → ⟨sloveso⟩ ⟨predmetová časť⟩
5. ⟨sloveso⟩ → ČÍTA
6. ⟨predmetová časť⟩ → ⟨podstatné meno⟩
7. ⟨podstatné meno⟩ → ČASOPIS

Na prvý pohľad by sa mohlo zdať, že prirodzený jazyk možno veľmi jednoducho formalizovať. Opak je však pravdou. Stačí si zobrať napr. vetu „Barbara číta knihu". Predmetovú časť vety už nestačí definovať pomocou jednotky ⟨podstatné meno⟩, ale bolo by potrebné uvažovať o páde podstatného mena. Problémov je však viac, no nebudeme sa nimi zaoberať.

Množina všetkých gramatických pravidiel jazyka tvorí *gramatiku jazyka*. Gramatika jazyka nám umožňuje generovať vety jazyka, ktoré majú základnú vlastnosť: sú gramaticky správne.

Gramatickými pravidlami — gramatikou — určujeme iba *syntax jazyka*, to znamená, že určujeme prípustnú štruktúru viet jazyka a prvotné jednotky, ktoré možno na danom mieste použiť. Ak ľubovoľná veta spĺňa podmienky definované gramatikou, patrí do daného jazyka. Význam viet určuje *sémantika*. Syntax a sémantika navzájom súvisia. Syntax definuje štruktúru vety, ktorá je zasa základom pri určovaní jej významu.

Syntaktická správnosť vety nezaručuje jej sémantickú správnosť. Možno to ilustrovať aj na gramatike vety prirodzeného jazyka, na základe ktorej možno vytvoriť napr. vety BARBARA ČÍTA ČASOPIS a ČASOPIS ČÍTA BARBARA. Ak budeme v predmetovej časti uvádzať predmet vo štvrtom páde, dostaneme dokonca vety ako BARBARA ČÍTA KNIHU, KNIHA ČÍTA BARBARU.

Vrátime sa teraz k formálnym jazykom a gramatikám. Navyše sa sústredíme na syntaktickú stránku jazyka.

2.1 GRAMATIKA

Aj pri formálnych jazykoch a gramatikách budeme pracovať s objektmi, ku ktorým sme prišli pri analýze prirodzeného jazyka: s prvotnými symbolmi, ktoré budeme nazývať *terminálne symboly* alebo jednoducho *terminály*; s jednotkami definovanými pomocou iných jednotiek, ktoré budeme nazývať *neterminálne symboly* alebo jednoducho *neterminály*; s gramatickými pravidlami, ktoré budeme nazývať *prepisovacie* (*vytvárajúce*) *pravidlá*.

Okrem uvedených objektov vystupuje v gramatike ešte jeden neterminálny symbol, ktorý má určité významné postavenie, pretože z neho sa generovanie viet začína. Preto ho aj nazývame *začiatočný symbol*. V analýze prirodzeného jazyka bola začiatočným symbolom ⟨veta⟩. Začiatočný symbol je neterminálny symbol.

Definícia 2.1. *Gramatikou* budeme nazývať štvoricu $G = (N, T, P, S)$, kde
N je konečná množina neterminálnych sybolov,
T — konečná množina terminálnych symbolov, pričom $N \cap T = \emptyset$,
S — začiatočný symbol, $S \in N$,
P — množina prepisovacích pravidiel, ktorá je konečnou podmnožinou množiny $(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$

Už pri prirodzených jazykoch sme hovorili, že na rozlíšenie neterminálnych a terminálnych symbolov používame špeciálny zápis — uzatváranie neterminálnych symbolov do lomených zátvoriek ⟨, ⟩. Aby sme zjednodušili zápis a zvýšili jeho čitateľnosť, v ďalšom texte budeme používať nasledujúci zápis:

a) neterminálne symboly budeme označovať veľkými latinskými písmenami,
b) terminálne symboly budeme označovať malými latinskými písmenami,
c) slová z neterminálnych a terminálnych symbolov budeme označovať malými gréckymi písmenami, slová iba z terminálnych symbolov malými latinskými písmenami s upozornením, že ide o reťazec (ak to nie je jasné z kontextu),
d) prepisovacie pravidlá budeme písať v tvare $\alpha \to \beta$,
e) prepisovacie pravidlá, ktoré majú rovnakú ľavú stranu, teda prepisovacie pravidlá tvaru

$$\alpha \to \beta_1$$
$$\alpha \to \beta_2$$
$$\vdots$$
$$\alpha \to \beta_n$$

budeme stručne písať ako

$$\alpha \to \beta_1 | \beta_2 | \dots | \beta_n$$

Tento zápis zodpovedá v podstate jazyku BNF (*Backusova—Naurova forma*), v ktorom autori *Backus* a *Naur* definovali programovací jazyk algol. BNF používa namiesto → symbol :: = a neterminálne symboly dáva do lomených zatvoriek ⟨, ⟩.

Príklad 2.1.

$G = (\{A, B\}, \{a, b\}, P, A)$
$P: A \to aAb \mid e$
$B \to bbA \mid aaB \mid a \mid B$
$aA \to bB \mid b$

Príklad 2.2.

$G = (\{I, R, C\}, \{a, b, c, 0, 1, 2\}, P, I)$
$P: I \to IR \mid IC \mid R$
$R \to a \mid b \mid c$
$C \to 0 \mid 1 \mid 2$

Gramatika sama ešte neposkytuje spôsob, ako z nej získať slová špecifikovaného jazyka. Základom pre generovanie viet jazyka sú prepisovacie pravidlá, na základe ktorých nad množinou reťazcov vytvorených z množiny neterminálnych a terminálovych symbolov definujeme *reláciu derivácie* alebo *reláciu odvodenia*.

Definícia 2.2. Nech $G = (N, T, P, S)$ je gramatika. Nad množinou $(N \cup T)^*$ potom definujeme *reláciu derivácie* $\underset{G}{\Rightarrow}$ nasledujúcim spôsobom: Ak $\alpha\beta\gamma \in (N \cup T)^*$ a $\alpha\delta\gamma \in (N \cup T)^*$, tak dané reťazce sú v relácii $\underset{G}{\Rightarrow}$, t. j. $\alpha\beta\gamma \underset{G}{\Rightarrow} \alpha\delta\gamma$, vtedy, ak v P existuje prepisovacie pravidlo $\beta \to \delta$.

Ak α, β sú reťazce z $(N \cup T)^*$ danej gramatiky G a platí $\alpha \underset{G}{\Rightarrow} \beta$, hovoríme, že β možno derivovať z α, alebo že z α možno priamo derivovať β.

Inverznú reláciu k relácii derivácie nazývame *redukcia*. Ak z α možno priamo derivovať β, tak tiež hovoríme, že β možno priamo redukovať na α.

Deriváciou reťazca dostávame ďalší reťazec. Je len prirodzené, že má zmysel hovoriť o tranzitívnosti a o mocnine relácie derivácie.

Definícia 2.3. Nech $G = (N, T, P, S)$ a $\alpha_i \in (N \cup T)^*$ pre $i = 0, 1, 2, \dots, k$. Potom hovoríme, že α_k sa dá derivovať z α_0, ak platí $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ pre $i = 0, 1, \dots$ $\dots, k - 1$. Postupnosť $\alpha_0, \alpha_1, \dots, \alpha_k$ nazývame *derivácia* α_k z α_0. Vyjadrovať ju budeme pomocou mocniny relácie $\underset{G}{\Rightarrow}$ v tvare $\alpha_0 \underset{G}{\Rightarrow}^k \alpha_k$.

Príklad 2.3. Nech $G = (\{A, B, C\}, \{0, 1\}, P, A)$

$$P: A \to BC1 \mid C0 \mid 0$$
$$B \to C1 \mid C0 \mid 1$$
$$C \to 0 \mid 1 \mid e$$

Ako príklady derivácie v gramatike G možno uviesť:

$ABC101AB \Rightarrow BC1BC101AB$ priama derivácia pomocou $A \to BC1$
$A \Rightarrow BC1$ priama derivácia pomocou $A \to BC1$
$AC1 \Rightarrow A1$ priama derivácia pomocou $C \to e$
$A \Rightarrow^4 111$ derivácia: $A \Rightarrow BC1 \Rightarrow C1C1 \Rightarrow 11C1 \Rightarrow 111$

Tranzitívny uzáver relácie $\underset{G}{\Rightarrow}$ budeme označovať $\underset{G}{\Rightarrow}^+$, reflexívny a tranzitívny uzáver $\underset{G}{\Rightarrow}^*$. $\alpha \underset{G}{\Rightarrow}^+ \beta$ platí práve vtedy, ak platí $\alpha \underset{G}{\Rightarrow}^n \beta$ pre $n \geqq 1$; $\alpha \underset{G}{\Rightarrow}^* \beta$ platí práve vtedy, ak platí $\alpha \underset{G}{\Rightarrow}^n \beta$ pre $n \geqq 0$.

Relácia derivácie nám umožňuje definovať významnú podmnožinu množiny $(N \cup T)^*$ — množinu vetných foriem alebo tvarov.

Definícia 2.4. Nech $G = (N, T, P, S)$ je gramatika. Množinu reťazcov $\alpha \in (N \cup T)^*$, ktoré sú derivovateľné zo začiatočného symbolu S, nazývame *množina vetných foriem.*

Množinu vetných foriem V možno formálne definovať takto:

$$V = \{\alpha / S \Rightarrow^* \alpha, \ \alpha \in (N \cup T)^*\}$$

Príklad 2.4. Majme gramatiku $G = (\{I, P, C\}, \{a, b, 0, 1\}, R, I)$

$$R: I \to P \mid IP \mid IC$$
$$P \to a \mid b$$
$$C \to 0 \mid 1$$

Z derivácie

$$I \Rightarrow IP \Rightarrow ICP \Rightarrow IPCP \Rightarrow PPCP \Rightarrow aPCP \Rightarrow abCP \Rightarrow ab1P \Rightarrow ab1a$$

možno potom vybrať nasledujúce vetné formy:

$I, IP, ICP,$
$IPCP, PPCP$
$aPCP, abCP$
$ab1P, ab1a$

Definícia 2.5. Nech $G = (N, P, T, S)$ je gramatika. Jazykom $L(G)$, *špecifikovaným gramatikou* G, nazývame množinu reťazcov pozostávajúcich z terminálnych symbolov, pre ktoré existuje derivácia zo začiatočného symbolu gramatiky S, formálne

$$L(G) = \{w / S \underset{G}{\Rightarrow}^* w, \ w \in T^*\}$$

Reťazec patriaci do daného jazyka budeme nazývať *veta jazyka.* Znova zjednodušíme zápis a ak bude z kontextu jasné, o ktorú gramatiku ide, budeme písmeno G v relácii derivácie vynechávať, teda namiesto $\underset{G}{\Rightarrow}$ budeme písať iba $\Rightarrow$.

Ak porovnáme predchádzajúce dve definície, vidíme, že vetná forma pozostávajúca iba z terminálnych symbolov je veta daného jazyka. Daný jazyk nemusí byť bezpodmienečne špecifikovaný iba jednou gramatikou, skôr naopak. Takýchto gramatík môže byť viac.

Definícia 2.6. Gramatiky G_1, G_2, pre ktoré platí $L(G_1) = L(G_2)$, nazývame *ekvivalentné.*

Príklad 2.5. Dané sú gramatiky $G_1 = (\{I, P, C\}, \{a, b, 0, 1\}, P_1, I)$

$$P_1: I \to P \mid IP \mid IC$$
$$P \to a \mid b$$
$$C \to 0 \mid 1$$
$$G_2 = (\{I, R\}, \{a, b, 0, 1\}, P_2, I)$$
$$P_2: I \to a \mid b \mid aR \mid bR$$
$$R \to a \mid b \mid 0 \mid 1 \mid aR \mid bR \mid 0R \mid 1R$$

Obidve uvedené gramatiky špecifikujú jazyk „obmedzených" identifikátorov, teda reťazcov nad abecedou $\{a, b, 0, 1\}$, ktoré však začínajú písmenom a alebo b. Preto $L(G_1) = L(G_2)$ a G_1 a G_2 sú ekvivalentné. Napríklad reťazec $abba1$ možno generovať nasledujúcim spôsobom.

$$G_1: I \Rightarrow IC \Rightarrow I1 \Rightarrow IP1 \Rightarrow Ia1 \Rightarrow IPa1 \Rightarrow Iba1 \Rightarrow IPba1 \Rightarrow Ibba1 \Rightarrow Pbba1 \Rightarrow abba1$$

$$G_2: I \Rightarrow aR \Rightarrow abR \Rightarrow abbR \Rightarrow abbaR \Rightarrow abba1$$

Možnosť generovať daný jazyk viacerými ekvivalentnými gramatikami je motiváciou pre skúmanie gramatík nielen z hľadiska toho, aký jazyk opisujú, ale aj ako ho opisujú, pre definovanie určitých vlastností gramatík a pre ich klasifikovanie. Najprv však budeme skúmať gramatiky z hľadiska jazyka, ktorý opisujú. Vhodným obmedzením prepisovacích pravidiel možno definovať určité hierarchické triedy jazykov a gramatík.

2.2 KLASIFIKÁCIA GRAMATÍK

Klasifikácia gramatík, ktorú v tomto článku urobíme, pochádza od *Chomského* a je založená na tvare prepisovacích pravidiel.

Definícia 2.7. Nech $G = (N, T, P, S)$ je gramatika. Potom hovoríme, že G je

a) *neobmedzená* alebo typu 0, ak na prepisovacie pravidlá nekladieme žiadne obmedzenie,

b) *kontextová* alebo typu 1, ak každé prepisovacie pravidlo z P má tvar
$\alpha \to \beta$,
kde $\alpha \in (N \cup T)^* N (N \cup T)^*$
$\beta \in (N \cup T)^+$
a platí $|\alpha| \leqq |\beta|$[1],

c) *bezkontextová* alebo typu 2, ak každé prepisovacie pravidlo z P má tvar
$A \to \alpha$, kde $A \in N$, $\alpha \in (N \cup T)^*$,

d) *regulárna* alebo typu 3, ak každé prepisovacie pravidlo z P má jeden z tvarov
$A \to bB$
$A \to b$
kde $A \in N$, $B \in N$, $b \in T$.

Niekedy je potrebné špecifikovať jazyk $L \cup \{e\}$, teda jazyk s prázdnym slovom, aj pre jazyky špecifikované typmi gramatík, v ktorých sme nepripustili použitie e na pravej strane pravidiel (regulárne a kontextové). Tento problém sa zvykne riešiť použitím rozšírenej gramatiky, ktorú k danej gramatike zostrojíme pridaním začiatočného symbolu S' a prepisovacích pravidiel $S' \to S$ a $S' \to e$. V ďalšom tak urobíme vždy, keď to budeme potrebovať, a nebudeme pritom explicitne upozorňovať, že ide o rozšírenú gramatiku. V tomto zmysle, ak jazyk L je kontextový, bezkontextový alebo regulárny, aj jazyky $L \cup \{e\}$ a $L - \{e\}$ sú kontextové, bezkontextové alebo regulárne.

Vzťahy gramatík podľa uvedenej klasifikácie sú nasledujúce:
a) každá regulárna gramatika je bezkontextová,
b) každá bezkontextová gramatika je kontextová,
c) každá kontextová gramatika je gramatika bez obmedzení.

Definícia 2.8. *Jazyk nazývame* postupne *regulárny*, *bezkontextový*, *kontextový* alebo *bez obmedzení*, ak ho možno generovať regulárnou, bezkontextovou, kontextovou gramatikou alebo gramatikou bez obmedzení.

Príklad 2.6.
Gramatika $G = (\{S\}, \{0,1\}, P, S)$
$P: S \to 0S \mid 1S \mid 0 \mid 1$ je regulárna
$L(G) = \{0, 1\}^+$
Gramatika $G = (\{E, T, F\}, \{a, +, \times, (,)\}, P, E)$
$P: E \to E + T \mid T$
$T \to T \times F \mid F$
$F \to (E) \mid a$

[1] Podmienky pre kontextovú gramatiku sa niekedy formulujú tak, že každé prepisovacie pravidlo musí mať tvar $\alpha A \gamma \to \alpha\beta\gamma$, kde $A \in N$, $\beta \in (N \cup T)^+$ a α, $\gamma \in (N \cup T)^*$, pričom α a γ nie sú súčasne prázdne reťazce. Dostaneme síce inú triedu gramatík, ale trieda jazykov, ktorú sme schopní opísať, zostane rovnaká.

je bezkontextová. Generovaným jazykom sú všetky aritmetické výrazy vytvorené z operátorov + a ×, zo zátvoriek a z operandov „a".

Gramatika $G = (\{S, A\}, \{0, 1\}, P, S)$
$P: S \to 0A1 \mid 01$
$0A \to 00A11 \mid 001$
je kontextová, generuje jazyk $\{0^n 1^n / n \geqq 1\}$.

Gramatika $G = (\{S, A\}, \{0, 1\}, P, S)$
$P: S \to 0A1$
$0A \to 00A1$
$A \to e$
je bez obmedzení vzhľadom na posledné prepisovacie pravidlo, kde neplatí podmienka $|\alpha| \leqq |\beta|$ pre prepisovacie pravidlo $\alpha \to \beta$, tu $A \to e$. Generuje jazyk $\{0^n 1^n / n \geqq 1\}$.

Príklad 2.7. Daná je gramatika $G = (\{S\}, \{a, +\}, P, S)$
$P: S \to S + S \mid a$

Vidíme, že gramatika G je bezkontextová. Jazyk $L(G)$, ktorý generuje, je množina reťazcov pozostávajúcich z a, za ktorým môže nasledovať ľubovoľný počet reťazcov $+ a$. Teda $L(G) = \{a, a + a, a + a + a, \ldots\}$. Túto množinu reťazcov však možno generovať aj nasledujúcou gramatikou G_1:
$G_1 = (\{S, A, B, C\}, \{a, +\}, P_1, S)$
$P_1: S \to a \mid aA$
$A \to +B \mid +C$
$B \to a$
$C \to aA$

Gramatiky G a G_1 sú ekvivalentné, $L(G) = L(G_1)$. Keďže však gramatika G_1 je regulárna, je regulárny jazyk $L(G_1)$, a teda aj $L(G)$.

Ak $\mathscr{L}_0$, $\mathscr{L}_1$, $\mathscr{L}_2$, $\mathscr{L}_3$ sú postupne triedy jazykov bez obmedzení, kontextových, bezkontextových a regulárnych, sú navzájom viazané nasledujúcim spôsobom:

$$\mathscr{L}_3 \subseteqq \mathscr{L}_2 \subseteqq \mathscr{L}_1 \subseteqq \mathscr{L}_0$$

V ďalších kapitolách sa zameriame iba na bezkontextové gramatiky a na ich podtriedu, regulárne gramatiky. Pri práci s programovacími jazykmi s nimi vystačíme. S najjednoduchšou triedou gramatík — regulárnymi gramatikami — vystačíme pri špecifikovaní základných objektov, identifikátorov, čísel a pod., zatiaľ čo zložitejšie konštrukcie, ako sú výrazy a príkazy, možno špecifikovať bezkontextovými gramatikami. Začneme regulárnymi gramatikami, ktoré dáme do súvisu s druhým opisovacím prostriedkom, automatom.

2.3 ZÁPIS SYNTAXE

Pre zápis syntaxe jazyka, t. j. množiny prepisovacích pravidiel určujúcich, ktoré reťazce vytvorené zo symbolov abecedy jazyka tvoria správne vytvorenú jazykovú konštrukciu, vetu jazyka, sa v súčasnosti najviac používajú dva metajazyky:

— BNF (Backusova—Naurova forma),

— syntaktický diagram alebo graf (niekedy aj Conwayove diagramy).

Jazyk BNF je špecifikovaný súborom prepisovacích pravidiel tvaru

⟨neterminálny symbol⟩ :: = $\alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

kde každé α_i na pravej strane reprezentuje reťazec neterminálnych a terminálnych symbolov alebo prázdny reťazec, pomocou ktorých je neterminálny symbol na ľavej strane prepisovacieho pravidla definovaný. Sám neterminálny symbol na ľavej strane reprezentuje syntaktickú jednotku, ktorú definujeme.

Zápis, ktorý sme doteraz používali, je teda v podstate jazyk BNF, ale namiesto :: = sme používali →. V súčasnosti sa často používa rozšírený jazyk BNF, v skratke EBNF, ktorý používa zložené zátvorky {, } s nasledujúcim účinkom:

Zápis

$\{\alpha\}$

znamená, že výskyt výrazu α sa na danom mieste môže opakovať ľubovoľný početkrát vrátane nulového. Uvedená konštrukcia sa používa ešte v ďalších modifikáciách. Zápis

$\{\alpha\}_{n_1}^{n_2}$

znamená, že výskyt výrazu α sa na danom mieste môže opakovať od n_1 po n_2-krát. Zápis

$\{\alpha\}^1$ alebo $[\alpha]$

znamená, že výraz α sa na danom mieste môže vyskytovať najviac jedenkrát.

Príklad 2.8. Identifikátor ako postupnosť písmen a číslic, ktorá sa začína písmenom, možno špecifikovať takto:

⟨identifikátor⟩ :: = ⟨písmeno⟩{⟨písmeno⟩ | ⟨číslica⟩}

⟨písmeno⟩ :: = A | B | ... | Z
⟨číslica⟩ :: = 0 | 1 | ... | 9

V mnohých implementáciách programovacích jazykov je však dĺžka identifikátora obmedzená maximálne na 6 znakov. Obmedzený identifikátor možno špecifikovať takto:

⟨identifikátor⟩ :: = ⟨písmeno⟩$\{$⟨písmeno⟩ | ⟨číslica⟩$\}_0^5$
⟨písmeno⟩ :: = A | B | ... | Z
⟨číslica⟩ :: = 0 | 1 | ... | 9

Celé číslo možno špecifikovať takto:

⟨celé číslo⟩ :: = $\{$⟨znamienko⟩$\}^1$⟨číslica⟩{⟨číslica⟩}
⟨znamienko⟩:: = + | −
⟨číslica⟩ :: = 0 | 1 | ... | 9

V poslednom čase, najmä v spojitosti s jazykom pascal, sa na zápis syntaxe používa *syntaktický diagram*. Je to v podstate grafické vyjadrenie väzieb medzi jednotlivými neterminálnymi a terminálnymi symbolmi gramatiky jazyka, určené prepisovacími pravidlami. Ide o vrcholovo ohodnotený orientovaný graf, v ktorom sú vrcholy grafu ohodnotené neterminálnymi a terminálnymi symbolmi gramatiky a hrany vyjadrujú väzby medzi nimi. Pre danú syntax jazyka ho možno vytvoriť nasledujúcim spôsobom:

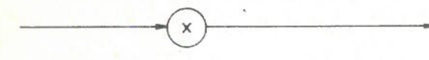
1. Každý neterminálny symbol A definovaný prepisovacími pravidlami

$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

sa zobrazí do syntaktického diagramu A podľa pravidiel 2 až 5. Štruktúra syntaktického diagramu je určená pravou stranou prepisovacieho pravidla.

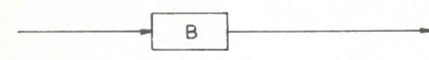
2. Každému výskytu terminálneho alebo neterminálneho symbolu v α_i zodpovedá v syntaktickom diagrame vrchol ohodnotený daným symbolom. Zobrazujú sa takto:

a) terminálny symbol x ako



a

b) neterminálny symbol B ako



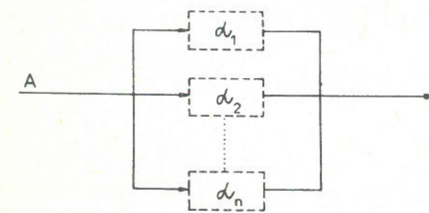
b

pričom → vyjadruje väzbu medzi symbolmi.

3. Prepisovacie pravidlá tvaru

$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

sa zobrazia do syntaktického diagramu



c

Obr. 2.2. Syntaktický diagram: a) pre terminálny symbol, b) pre neterminálny symbol, c) pre predpisovacie pravidlá $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

pričom každé α_i, $i = 1, 2, \ldots, n$, možno získať pomocou pravidiel 2 až 5. To znamená, že ak je to neterminálny symbol, zodpovedajú mu prepisovacie pravidlá, ktorými je daný symbol definovaný, a podľa ich pravých strán preň vytvoríme syntaktický diagram. Ak je to terminálny symbol, netreba robiť nič — je definovaný svojím výskytom. Nakoniec, ak je to reťazec, rozložíme ho podľa pravidiel 2 až 5.

4. Ak má α_i tvar

$$\alpha_i = x_1 x_2 \ldots x_m$$

zobrazí sa do diagramu (*obr. 2.3*), pričom každé x_i, $i = 1, 2, \ldots, m$, možno získať pomocou pravidiel 2 až 5.

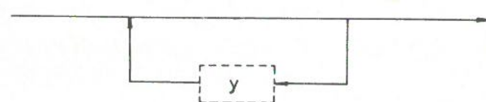


Obr. 2.3. Syntaktický diagram pre $\alpha = x_1 x_2 \ldots x_m$

5. Ak má α_i tvar

$$\alpha_i = \{y\}$$

zobrazí sa do diagramu (*obr. 2.4*), pričom y možno získať pomocou pravidiel 2 až 5.

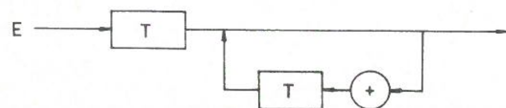


Obr. 2.4. Syntaktický diagram pre reťazec $\alpha = \{y\}$

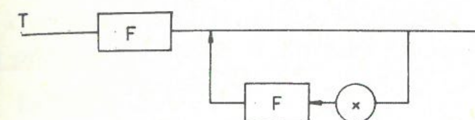
Príklad 2.9. Dané sú tieto prepisovacie pravidlá:

$E \to T\{+ T\}$
$T \to F\{\times F\}$
$F \to (E) \mid a$

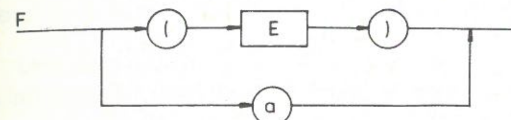
Možno ich zobraziť nasledujúcimi syntaktickými diagramami (*obr. 2.5, 2.6, 2.7*):



Obr. 2.5. Syntaktický diagram pre prepisovacie pravidlo $E \to T\{+ T\}$



Obr. 2.6. Syntaktický diagram pre prepisovacie pravidlo $T \to F\{\times F\}$



Obr. 2.7. Syntaktický diagram pre prepisovacie pravidlá $F \to (E) \mid a$

Cvičenia

1. Vytvorte gramatiku a zodpovedajúci syntaktický diagram špecifikujúci prirodzené čísla. Akého typu je gramatika, ktorú ste vytvorili? Akého typu je špecifikovaný jazyk?

2. Vytvorte gramatiku a zodpovedajúci syntaktický diagram špecifikujúci:
a) celé čísla,
b) reálne čísla v desatinnom a semilogaritmickom tvare,
c) identifikátory ako postupnosti písmen a čísel, ktoré začínajú písmenom, resp. obmedzenou množinou písmen,
d) zoznam,
e) aritmetický výraz pozostávajúci z premenných i, j čísel 0, 1, operátorov $+, -, \times, /, \uparrow$ (umocnenie) a zátvoriek(,).
Urobte analýzu vytvorených gramatík a zodpovedajúcich jazykov ako v cvičení 1.

3. V literatúre sa často uvádza špecifikácia regulárneho jazyka *vpravo lineárnou gramatikou*. Gramatika $G = (N, T, P, S)$ je *vpravo lineárna*, ak každé prepisovacie pravidlo má jeden z tvarov

$A \to uB$
$A \to u$

kde $A, B \in N$
$u \in T^*$

a) Daná je vpravo lineárna gramatika

$G = (\{S, A, B\}, \{+, -, 12, 14, .\}, P, S)$
$P: S \to 12A \mid 14A \mid 12B \mid 14B$
$A \to .$
$B \to . + S \mid . - S$

Vytvorte pre G ekvivalentnú regulárnu gramatiku.

b) Možno ku každej vpravo lineárnej gramatike vytvoriť ekvivalentnú regulárnu gramatiku? Ak áno, dokážte to.

3 REGULÁRNE JAZYKY

V predchádzajúcej kapitole sme sa zaoberali prostriedkami špecifikácie jazyka generatívnym spôsobom. Ukázali sme, ako generovať (gramaticky správne) vety daného jazyka. V tejto kapitole sa budeme zaoberať prostriedkami špecifikácie jazyka akceptačným spôsobom pomocou automatov. Ukážeme, ako možno zistiť, či nejaký reťazec patrí alebo nepatrí do daného jazyka. Naviac ukážeme, ako možno vytvoriť prostriedok, automat, ktorý bude slúžiť na špecifikáciu jazykov daného typu.

Je dosť prirodzené, že zložitosť procesu rozhodovania, či daný reťazec patrí alebo nepatrí do jazyka, závisí od typu jazyka. Začneme preto od najjednoduchších prostriedkov — konečných automatov (KA).

3.1 KONEČNÝ AUTOMAT

V tomto článku zavedieme pojem konečného automatu, ukážeme spôsob jeho práce i formy zápisu. Podľa spôsobu práce rozdelíme konečné automaty na deterministické a nedeterministické.

Definícia 3.1. Pod *deterministickým konečným automatom M* rozumieme päticu

$M = (Q, T, \delta, q_0, F)$

kde Q je konečná množina vnútorných stavov automatu,
T — konečná množina prípustných vstupných symbolov,
δ — prechodové zobrazenie $\delta: Q \times T \rightarrow Q$, ktoré nazývame *prechodová funkcia*,
q_0 — začiatočný stav, $q_0 \in Q$,
F — množina koncových stavov, $F \subseteq Q$.

Konečný automat pracuje diskrétne po určitých krokoch alebo taktoch. V každom kroku načíta jeden symbol vstupného reťazca, ktorý spolu so stavom,

v ktorom sa práve nachádza, rozhodne o ďalšej činnosti automatu. Táto činnosť je pre daný automat predpísaná jeho prechodovým zobrazením δ.

Definícia 3.2. Nech $M = (Q, T, \delta, q_0, F)$ je konečný automat. Dvojicu $(q, w) \in Q \times T^*$ nazývame *konfigurácia konečného automatu M*. Konfiguráciu (q_0, w) kde w je vstupný reťazec, nazývame *začiatočná konfigurácia automatu M*, konfiguráciu (q, e), kde $q \in F$, *koncová konfigurácia automatu M*.

Definícia 3.3. Nech $M = (Q, T, \delta, q_0, F)$ je konečný automat. Nad množinou konfigurácií $Q \times T^*$ definujeme *reláciu prechodu* $\vdash_M$ takto:

Ak

$q \in Q,\ p \in Q,\ w \in T^*,\ a \in T$

tak

$(q, aw) \vdash_M (p, w)$, ak $\delta(q, a) = p$

V označení relácie prechodu $\vdash_M$ budeme M vynechávať, ak bude jasné, pre ktorý automat je relácia definovaná. *k-tú mocninu relácie* $\vdash$ budeme označovať $\overset{k}{\vdash}$, *tranzitívny uzáver* $\vdash^+$, *reflexívny a tranzitívny uzáver* $\vdash^*$.

Definícia 3.4. Nech $M = (Q, T, \delta, q_0, F)$ je konečný automat. Potom hovoríme, že stav $q \in Q$ je *dosiahnuteľný*, ak existuje prechod $(q_0, w) \overset{n}{\vdash} (q, e)$ pre nejaké $n \geqq 0$, $w \in T^*$. Ak taký prechod neexistuje, stav q je *nedosiahnuteľný*.

Dosiahnuteľný stav je teda taký, do ktorého sa môže automat dostať z niektorej začiatočnej konfigurácie po nejakom počte prechodov.

Definícia 3.5. Nech $M = (Q, T, \delta, q_0, F)$ je konečný automat. Potom hovoríme, že automat M prijíma (akceptuje) reťazec $w \in T^*$, ak platí

$(q_0, w) \vdash^* (q, e)$ pre nejaké $q \in F$

Automat môže tým, že niektoré reťazce akceptuje a niektoré nie, slúžiť ako prostriedok na špecifikáciu jazyka.

Definícia 3.6. Ak $M = (Q, T, \delta, q_0, F)$ je konečný automat, tak *jazyk L(M) špecifikovaný automatom M* je vyjadrený takto:

$L(M) = \{w / (q_0, w) \vdash^* (q, e), w \in T^*, q \in F\}$

Jazyk špecifikovaný automatom je teda taká množina reťazcov zo vstupnej abecedy, ktoré sú prijaté, akceptované daným automatom.

Príklad 3.1. Daný je konečný automat

$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, q_0, \delta, \{q_0\})$
δ: $\delta(q_0, 0) = q_2$ $\quad$ $\delta(q_0, 1) = q_1$
$\delta(q_1, 0) = q_3$ $\quad$ $\delta(q_1, 1) = q_0$
$\delta(q_2, 0) = q_0$ $\quad$ $\delta(q_2, 1) = q_3$
$\delta(q_3, 0) = q_1$ $\quad$ $\delta(q_3, 1) = q_2$

Prijatie reťazca 10101010 sa uskutočňuje nasledujúcim spôsobom:

$(q_0, 10101010) \vdash (q_1, 0101010) \vdash (q_3, 101010) \vdash (q_2, 01010) \vdash (q_0, 1010) \vdash$
$\vdash (q_1, 010) \vdash (q_3, 10) \vdash (q_2, 0) \vdash (q_0, e)$

Doteraz sme sa zaoberali situáciami, ktoré boli „priaznivé" — postupným prechodom sme sa dostali do konfigurácie, ktorá bola koncová. V takomto prípade automat reťazec prijal a z jazykového hľadiska reťazec patril do daného jazyka. Situáciu, keď automat neakceptuje vstupný reťazec, rozpoznáme tak, že automat sa po dočítaní vstupného reťazca nenachádza v koncovej konfigurácii. V praktických aplikáciách môže takáto situácia nastať aj vtedy, keď vstupný reťazec obsahuje symbol, ktorý nie je z danej abecedy.

V prípade, že sa automat dostane do stavu, ktorý nie je koncový, ale vstup je už prázdny, alebo do stavu, z ktorého pre momentálny začiatočný symbol vstupného reťazca, resp. jeho nespracovanej časti zobrazenie δ nie je definované, automat sa zastaví neštadardným spôsobom a daný vstupný reťazec nie je prijatý, a teda nepatrí ani do daného jazyka.

Príklad 3.2. Majme znova automat z príkladu 3.1 a zistime jeho správanie pre vstupné reťazce 101010, 1012:

$(q_0, 101010) \vdash (q_1, 01010) \vdash (q_3, 1010) \vdash (q_2, 010) \vdash (q_0, 10) \vdash (q_1, 0) \vdash (q_3, e)$

Keďže na vstupe je prázdny reťazec a q_3 nie je koncovým stavom, automat sa zastaví neštandardným spôsobom a reťazec 101010 nepatrí do $L(M)$.

$(q_0, 1012) \vdash (q_1, 012) \vdash (q_3, 12) \vdash (q_2, 2)$

Keďže $\delta(q_2, 2)$ nie je definované, reťazec nepatrí do $L(M)$. (Je to jasné, pretože 2 nie je z T.)

Na reprezentáciu konečných automatov používame:
a) prechodovú tabuľku,
b) prechodový diagram.

Prechodová tabuľka P reprezentuje prechodové zobrazenie δ. Ak je množina stavov $Q = \{q_0, q_1, \ldots, q_n\}$ a množina vstupných symbolov $T = \{t_0, t_1, \ldots, t_k\}$, prechodová tabuľka P má n riadkov a k stĺpcov, zodpovedajúcich Q a T, a prvok $P[i, j] = \delta(q_i, t_j)$.

Prechodový diagram je orientovaný graf, v ktorom vrcholy sú ohodnotené stavmi a hrany symbolmi vstupnej abecedy tak, že ak $\delta(q, t) = p$, tak existuje orientovaná hrana z q do p, ktorá je ohodnotená t. Začiatočný a koncový stav je vyznačený špeciálnym spôsobom: napr. začiatočný stav šípkou s označením ŠTART, koncový stav dvojitým krúžkom.

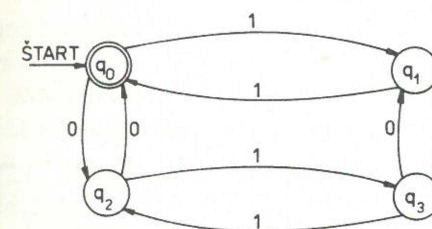
Činnosť KA reprezentovaného prechodovým diagramom sa začína vrcholom s označením ŠTART a postupne sa prechádza po hranách určených zodpovedajúcim symbolom zo vstupného reťazca. Reťazec je prijatý alebo odmietnutý za rovnakých podmienok ako pri priamom použití relácie prechodu.

Príklad 3.3. Majme znova automat z príkladu 3.1. Jeho reprezentáciu prechodovou tabuľkou potom môžeme napísať takto (*tab. 3.1*):

Tabuľka 3.1

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Reprezentácia prechodovým diagramom je na *obr. 3.1*:



Obr. 3.1. Prechodový diagram pre automat z príkladu 3.1

Všeobecnejším typom konečného automatu je nedeterministický KA. Jeho základnou vlastnosťou je, že momentálny stav a začiatočný symbol vstupného reťazca neurčujú jednoznačne stav, do ktorého automat môže prejsť. Oproti deterministickému KA sa však zmení iba prechodová funkcia.

Definícia 3.7. *Nedeterministickým konečným automatom M* nazývame pätícu

$M = (Q, T, \delta, q_0, F)$

kde Q je konečná množina vnútorných stavov automatu,
T — konečná množina vstupných symbolov — vstupná abeceda,

δ — prechodové zobrazenie $\delta: Q \times T \to 2^Q$,
$q \in Q$ — začiatočný stav,
F — množina koncových stavov, $F \subseteq Q$.

Činnosť nedeterministického KA budeme aj teraz definovať na základe konfigurácií. Význam konfigurácie, začiatočná a koncová konfigurácia zostávajú rovnaké ako pri deterministickom KA. Keďže sa však zmenilo prechodové zobrazenie, treba zmeniť i reláciu prechodu.

Definícia 3.8. Nech $M = (Q, T, \delta, q_0, F)$ je nedeterministický KA. Nad množinou konfigurácií $Q \times T^*$ definujeme *reláciu prechodu* $\underset{M}{\vdash}$ nasledujúcim spôsobom:

Ak $q \in Q$, $p \in Q$, $w \in T^*$, $a \in T$, tak

$(q, aw) \underset{M}{\vdash} (p, w)$, ak $p \in \delta(q, a)$

Prijatie reťazca nedeterministickým KA i jazyk špecifikovaný nedeterministickým KA sú definované rovnakým spôsobom ako pre deterministický KA, teda w je prijaté, ak

$(q_0, w) \vdash^* (q, e)$ pre nejaké $q \in F$

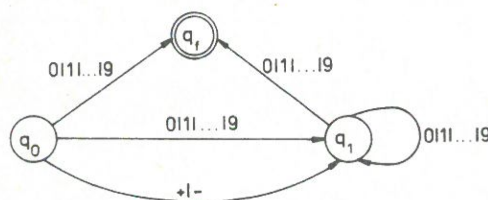
a

$L(M) = \{w / w \in T^*, (q_0, w) \vdash^* (q, e), q \in F\}$

Príklad 3.4. Daný je automat

$M = (\{q_0, q_1, q_f\}, \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \delta, q_0, \{q_f\})$

Reprezentovať ho budeme nasledujúcim prechodovým diagramom (*obr. 3.2*), v ktorom pre zjednodušenie budeme ohodnocovať hrany viacerými vstupnými symbolmi s alternatívnym výberom.



Obr. 3.2. Prechodový diagram pre automat z príkladu 3.4

Uvedený KA je nedeterministický, ako vyplýva zo stavového diagramu, v ktorom napr. zo stavu q_1 možno ísť dvoma cestami. Nedeterministickosť vidieť aj z prechodovej tabuľky. Pre M má tvar ako v *tab. 3.2.*

Tabuľka 3.2

δ	$+ \mid -$	$0 \mid 1 \mid \ldots \mid 9$
q_0	$\{q_1\}$	$\{q_1, q_f\}$
q_1		$\{q_1, q_f\}$
q_f		

Prvky prechodovej tabuľky, ktorými sú teraz množiny, dávajú viac alternatívnych možností prechodu. Pri prijímaní reťazca -192 existujú nasledujúce možnosti prechodov:

$(q_0, -192) \vdash (q_1, 192) \vdash (q_1, 92) \vdash (q_2, 2) \vdash (q_1, e)$
$(q_2, 2) \vdash (q_f, e)$
$(q_1, 92) \vdash (q_f, 2)$
$(q_1, 192) \vdash (q_f, 92)$

Ľahko možno vidieť, že jazyk špecifikovaný automatom M je množina celých čísel so znamienkom a pripúšťa nevýznamné nuly zľava.

Keďže máme definované dva typy konečných automatov, vzniká prirodzená otázka, aký je ich vzťah. Prijímajú rovnakú triedu jazykov? Odpoveď nám dáva nasledujúca veta.

Veta 3.1. Nech L je jazyk prijatý nejakým nedeterministickým konečným automatom. Potom existuje deterministický konečný automat, ktorý prijíma práve jazyk L.

Dôkaz. Urobíme konštrukciu deterministického KA k danému nedeterministickému KA, ktorý prijme práve ten istý jazyk ako pôvodný nedeterministický KA. Nech $M = (Q, T, \delta, q_0, F)$ je nedeterministický KA, ktorý prijíma množinu reťazcov L. Teraz skonštruujeme deterministický konečný automat $M' = (Q', T, \delta', q_0', F')$, a to nasledujúcim spôsobom: Stavy $q' \in Q'$ automatu M' budeme označovať $[q_1, q_2, \ldots, q_i]$, kde $q_k \in Q$ pre $k = 1, 2, \ldots, i$. Automat M' bude sledovať všetky stavy, v ktorých sa M môže nachádzať v ľubovoľnom kroku $q_0' = [q_0]$. Množina koncových stavov F' je tvorená množinou všetkých tých stavov z Q', ktoré obsahujú aspoň jeden koncový stav z automatu M.

Prechodové zobrazenie δ' budeme definovať ako

$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$

vtedy a len vtedy, ak

$\delta(\{q_1, \ldots, q_i\}, a) = \{p_1, \ldots, p_j\}$

To znamená, že δ' aplikované na prvok X množiny Q' vypočítame tak, že aplikujeme δ na každý stav z množiny Q obsiahnutý v stave $X = [q_1, q_2, \ldots, q_i]$.

Aplikovaním δ na každý zo stavov $q_1, q_2, \ldots, q_i$ a zjednotením výsledných množín dostaneme nejakú novú množinu stavov $\{p_1, p_2, \ldots, p_j\}$, reprezentujúcu v množine stavov Q' stav $[p_1, p_2, \ldots, p_j]$, ktorý je hodnotou $\delta'([q_1, q_2, \ldots, q_i], a)$.

Keďže každý stav automatu M' reprezentuje množinu stavov automatu M, budeme v ďalšom kvôli jednoduchosti používať tieto zápisy:

— ak $q' = [q_1, q_2 \ldots, q_n]$, tak $q_i \in q'$ pre $i = 1, \ldots, n$,

— $\delta'([q_1, \ldots, q_i], a) = \left[\bigcup_{j=1}^{i} \delta(q_j, a)\right]$.

Takto zostrojený automat je deterministický konečný automat, ktorého stavmi sú prvky z 2^Q a prechodové zobrazenie je

$$\delta': 2^Q \times T \to 2^Q$$

Teraz máme dokázať že $L(M) = L(M')$, to znamená, že ľubovoľný reťazec $u \in T^*$ bude prijatý automatom M' práve vtedy, ak bude prijatý automatom M, t. j.

$$(q_0', u) \vdash_{M'}^{*} (q', e) \text{ pre } q' \in F' \leftrightarrow (q_0, u) \vdash_{M}^{*} (q, e) \text{ pre } q \in F$$

Dôkaz. Urobíme ho indukciou vzhľadom na dĺžku prečítaného reťazca.

1. $k = 1$.

Podľa konštrukcie M' platí $q_0' = [q_0]$ a

$$\delta'([q_0], a) = [\delta(q_0, a)] \text{ pre všetky } a \in T$$

Teraz ak $(q_0', a) \vdash (q_f', e)$, tak q_f' musí obsahovať také q_f, že $q_f \in \delta(q_0, a)$, a preto platí aj

$$(q_0, a) \vdash (q_f, e)$$

Ak $(q_0, a) \vdash (q_f, e)$, tak $\delta(q_0, a)$ musí obsahovať q_f, a preto stav $[\delta(q_0, a)]$ v M' je koncový a platí aj

$$(q_0', a) \vdash (q_f', e)$$

2. Predpokladajme, že pre $1 \leqq k \leqq n$ platí

$$(q_0', a_1 a_2 \ldots a_n) \vdash_{M'}^{*} (q_{k-1}', a_k a_{k+1} \ldots a_n)$$

vtedy a len vtedy, ak

$$(q_0, a_1 a_2 \ldots a_n) \vdash_{M}^{*} (q_{k-1}, a_k a_{k+1} \ldots a_n)$$

Potom podľa konštrukcie M'

$$\delta'(q_{k-1}', a_k) = \left[\bigcup_{q \in q_{k-1}'} \delta(q, a_k)\right] = q_k'$$

Teraz ak

$$(q_{k-1}', a_k a_{k+1} \ldots a_n) \vdash_{M'} (q_k', a_{k+1} a_{k+2} \ldots a_n)$$

tak pre každé $q_k \in q_k'$ existuje také $q \in q_{k-1}'$, že $\delta(q, a_k)$ obsahuje q_k, a preto platí aj

$$(q_{k-1}, a_k a_{k+1} \ldots a_n) \vdash_{M} (q_k, a_{k+1} a_{k+2} \ldots a_n)$$

Nech teraz platí

$$(q_{k-1}, a_k a_{k+1} \ldots a_n) \vdash_{M} (q_k, a_{k+1} a_{k+2} \ldots a_n)$$

Potom ale $\delta(q_{k-1}, a_k)$ musí obsahovať q_k a každé $q_{k-1} \in q_{k-1}'$ je $q_k \in \delta'(q_{k-1}', a_k)$, a preto platí aj

$$(q_{k-1}', a_k a_{k-1} \ldots a_n) \vdash_{M'} (q_k', a_{k+1} a_{k+2} \ldots a_n)$$

Možno teda písať, že

$$(q_0', a_1 a_2 \ldots a_n) \vdash_{M'}^{*} (q_k', a_{k+1} a_{k+2} \ldots a_n) \text{ vteda a len vtedy, ak}$$

$$(q_0, a_1 a_2 \ldots a_n) \vdash_{M}^{*} (q_k, a_{k+1} a_{k+2} \ldots a_n)$$

Ak jazyk $L(M)$ obsahuje i prázdny reťazec, tak platí $q_0 \in F$, ako aj $q_0' = [q_0] \in F'$.

Výsledok, že deterministický aj nedeterministický konečný automat prijímajú alebo rozpoznávajú ten istý jazyk, má pre nás význam nielen z hľadiska existencie takých automatov, ale aj z hľadiska konštrukcie deterministického konečného automatu k nedeterministickému.

Príklad 3.5. Daný je automat $M = (\{q_0, q_1, q_2, q_f\}, \{0, 1\}, \delta, q_0, \{q_f\})$, kde prechodové zobrazenie je definované takto:

$$\delta(q_0, 0) = \{q_0, q_1\} \qquad \delta(q_0, 1) = \{q_0, q_2\}$$
$$\delta(q_1, 0) = \{q_1, q_f\} \qquad \delta(q_1, 1) = \{q_1\}$$
$$\delta(q_2, 0) = \{q_2\} \qquad \delta(q_2, 1) = \{q_2, q_f\}$$

Vidíme, že M je nedeterministický konečný automat. Zostrojíme k nemu deterministický konečný automat M', ktorý bude prijímať ten istý jazyk ako prijíma M. Podľa konštrukcie z vety 3.1 bude M' určený päticou

$$M' = (Q', \{0, 1\}, \delta', q_0', F'), \text{ pričom } q_0' = [q_0]$$

Prechodové zobrazenie δ' vytvoríme podľa konštrukcie z vety 3.1 takto:

$$\delta'([q_0], 0) = [\delta(q_0, 0)] = [q_0, q_1]$$
$$\delta'([q_0, q_1], 0) = [\delta(q_0, 0) \cup \delta(q_1, 0)] = [q_0, q_1, q_f]$$

Podobne by sme postupovali pre ostatné stavy. Prechodové zobrazenie δ' pre stavy, ku ktorým vedie nejaká postupnosť prechodov zo začiatočného stavu $[q_0]$, t. j. dosiahnuteľné stavy, sú v prechodovej tabuľke (*tab. 3.3*)

Tabuľka 3.3

δ'	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_f]$	$[q_0, q_1, q_2]$
$[q_0, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_2, q_f]$
$[q_0, q_1, q_f]$	$[q_0, q_1, q_f]$	$[q_0, q_1, q_2]$
$[q_0, q_2, q_f]$	$[q_0, q_1, q_2]$	$[q_0, q_2, q_f]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_f]$	$[q_0, q_1, q_2, q_f]$
$[q_0, q_1, q_2, q_f]$	$[q_0, q_1, q_2, q_f]$	$[q_0, q_1, q_2, q_f]$

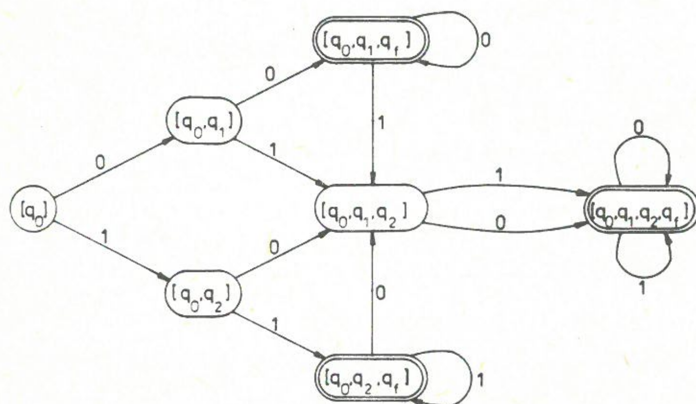
Z *tab. 3.3* vidieť, že

$$Q' = \{[q_0], [q_0, q_1], [q_0, q_2], [q_0, q_1, q_f], [q_0, q_2, q_f], [q_0, q_1, q_2], [q_0, q_1, q_2, q_f]\}$$
$$F' = \{[q_0, q_1, q_f], [q_0, q_2, q_f], [q_0, q_1, q_2, q_f]\}$$

Prijatie reťazca 111000 prebieha takto:

$$([q_0], 111000 \vdash ([q_0, q_2], 11000) \vdash ([q_0, q_2, q_f], 1000) \vdash ([q_0, q_2, q_f], 000) \vdash$$
$$\vdash ([q_0, q_1, q_2], 00) \vdash ([q_0, q_1, q_2, q_f], 0) \vdash ([q_0, q_1, q_2, q_f], e)$$

Keďže stav $[q_0, q_1, q_2, q_f]$ je jeden z koncových stavov, reťazec je prijatý. Prechodový diagram deterministického konečného automatu M je na *obr. 3.3.*



Obr. 3.3. Prechodový diagram deterministického automatu pre automat z príkladu 3.5

Z definície 3.4 vidieť, že automat môže obsahovať stavy, ktoré nie sú dosiahnuteľné zo začiatočného stavu, resp. zo začiatočnej konfigurácie. Pretože tieto stavy neovplyvňujú prijatie alebo neprijatie reťazca daným automatom, možno ich z množiny stavov vylúčiť a automat redukovať tak, aby prijímal rovnakú množinu reťazcov ako pôvodný automat. Dosahujeme to určitým rozkladom množiny dosiahnuteľných stavov.

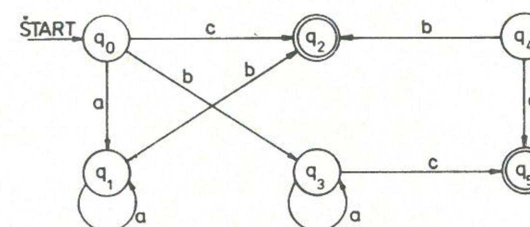
Definícia 3.9. Nech $M = (Q, T, \delta, q, F)$ je deterministický konečný automat a q_1, q_2 sú dva rôzne stavy. Budeme hovoriť, že:

— *reťazec u rozlišuje stavy* q_1 a q_2 vtedy, ak $(q_1, u) \vdash^* (q_3, e)$, $(q_2, u) \vdash^* (q_4, e)$ a jeden zo stavov q_3 alebo q_4 patrí do F, ale druhý nepatrí,

— q_1 a q_2 sú *k-nerozlíšiteľné* a budeme písať $q_1 \equiv^k q_2$, ak neexistuje reťazec $u \leq k$, ktorý q_1 a q_2 rozlišuje,

— stavy q_1 a q_2 sú nerozlíšiteľné a píšeme $q_1 \equiv q_2$, ak sú k-nerozlíšiteľné pre ľubovoľné $k \geq 0$. Automat M budeme nazývať redukovaným, ak v Q neexistujú nedosiahnuteľné stavy a neexistujú žiadne dva nerozlíšiteľné stavy.

Príklad 3.6. Majme automat reprezentovaný nasledujúcim prechodovým diagramom (*obr. 3.4*):



Obr. 3.4. Prechodový diagram pre automat z príkladu 3.6

V uvedenom automate sú stavy q_0 a q_3 1-nerozlíšiteľné, q_4 je nedosiahnuteľný, reťazec b rozlišuje stavy q_0 a q_1, stavy q_2 a q_5 sú 0-nerozlíšiteľné.

Lema 3.1. Nech $M = (Q, T, \delta, q_0, F)$ je deterministický konečný automat s n-stavmi. Stavy q_1 a q_2 sú potom nerozlíšiteľné vtedy a len vtedy, keď sú $(n-2)$-nerozlíšiteľné.

Dôkaz. Nutná podmienka je triviálna. Postačujúca podmienka je triviálna v prípade, že F má 0 alebo n stavov.

$$\equiv \subseteq \equiv^{n-2} \subseteq \equiv^{n-3} \subseteq \ldots \subseteq \equiv^2 \subseteq \equiv^1 \subseteq \equiv^0$$

Ľahko vidieť, že pre ľubovoľné stavy q_1 a q_2 platí:

1. $q_1 \equiv^0 q_2$ vtedy a len vtedy, ak q_1 a q_2 oba patria alebo nepatria do F,

2. $q_1 \equiv^k q_2$ vtedy a len vtedy, ak $q_1 \equiv^{k-1} q_2$ a $\delta(q_1, a) \equiv^{k-1} \delta(q_2, a)$ pre všetky $a \in T$.

Relácia $\equiv^0$ rozdeľuje Q na dve triedy: F a $Q - F$. Ak $\equiv^{k+1} \neq \equiv^k$, tak relácia $\equiv^{k+1}$ je jemnejšia, teda obsahuje aspoň o jednu triedu ekvivalencie viac. Pretože žiadna z množín F, $Q - F$ neobsahuje viac ako $n - 1$ stavov, nemožno vytvoriť viac ako $n - 2$ postupných zjemnení relácie $\equiv^0$. Ak $\equiv^{k+1} = \equiv^k$ pre nejaké k, tak vzhľadom na bod 2 $\equiv^{k+1} = \equiv^{k+2} = \ldots$. Preto relácia $\equiv$ je prvou z relácií $\equiv^k$, pre $\equiv^{k+1} = \equiv^k$.

Redukovaný automat možno vytvoriť nasledujúcim algoritmom:

Algoritmus 3.1. Konštrukcia redukovaného automatu.

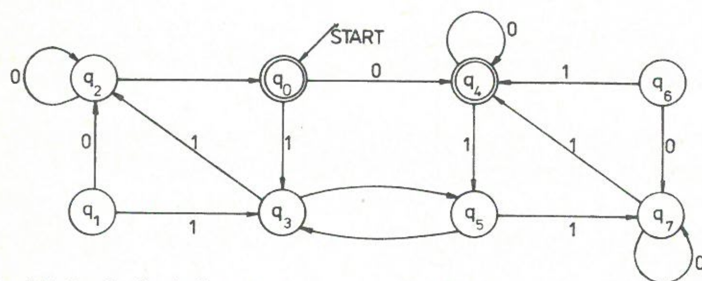
Vstup: Deterministický konečný automat $M = (Q, T, \delta, q_0, F)$.

Výstup: Ekvivalentný redukovaný konečný automat M'.

Metóda:

1. Nájdeme všetky nedosiahnuteľné stavy zo stavu q_0 a vylúčime ich z Q.
2. Podľa lemy 3.1 vytvárame relácie ekvivalencie $\equiv^0, \equiv^1, \ldots$, a to dovtedy, kým sa dve susedné nebudú rovnať, teda kým nebude platiť $\equiv^{k+1} = \equiv^k$. Potom za $\equiv$ zoberieme $\equiv^k$.
3. Vytvoríme konečný automat $M' = (Q', T, \delta', q_0', F')$:
   a) Q' je množina tried relácie ekvivalencie $\equiv$.
   b) Označme $[p]$ triedu relácie ekvivalencie $\equiv$ obsahujúcu stav p. Potom $\delta'([p], a) = [q]$, ak $\delta(p, a) = q$.
   c) $q_0' = [q_0]$.
   d) $F' = \{[q]/q \in F\}$.

Príklad 3.7. Majme automat reprezentovaný nasledujúcim prechodovým diagramom (*obr. 3.5*). Pomocou algoritmu 3.1 vytvoríme ekvivalentný redukovaný automat.



Obr. 3.5. Prechodový diagram pre automat z príkladu 3.7

Nedosiahnuteľné sú stavy q_1 a q_6, a preto ich vylúčime. Jednotlivé triedy ekvivalencie pre dosiahnuteľné stavy sú:

$\equiv^0$: $\{q_0, q_4\}, \{q_2, q_3, q_5, q_7\}$
$\equiv^1$: $\{q_0, q_4\}, \{q_2, q_7\}, \{q_3, q_5\}$
$\equiv^2$: $\{q_0, q_4\}, \{q_2, q_7\}, \{q_3, q_5\}$

Pretože $\equiv^2 = \equiv^1$, položíme $\equiv = \equiv^1$. Pre redukovaný automat M' možno teda písať

$Q' = \{[q_2], [q_0], [q_3]\}$

δ':

δ'	0	1
$[q_0]$	$[q_0]$	$[q_3]$
$[q_2]$	$[q_2]$	$[q_0]$
$[q_3]$	$[q_3]$	$[q_2]$

$q_0' = [q_0]$
$F = \{[q_0]\}$

Pozrime sa, ako prebieha proces rozpoznávania reťazca 100101 oboma automatmi.

M: $(q_0, 100101) \vdash (q_3, 00101) \vdash (q_5, 0101) \vdash (q_3, 101) \vdash (q_2, 01) \vdash$
$\vdash (q_2, 1) \vdash (q_0, e)$

M': $([q_0], 100101) \vdash ([q_3], 00101) \vdash ([q_3], 0101) \vdash$
$\vdash ([q_3], 101) \vdash ([q_2], 01) \vdash ([q_2], 1) \vdash ([q_0], e)$

Veta 3.2. Automat M' vytvorený na základe algoritmu 3.1 má najmenší počet stavov zo všetkých automatov špecifikujúcich jazyk $L(M)$.

Dôkaz. Predpokladajme, že tvrdenie vety neplatí a že existuje automat M'' s menším počtom stavov, ako má M', a že $L(M'') = L(M)$. Vzhľadom na krok 1 algoritmu 3.1 sú všetky stavy automatu M' dosiahnuteľné. Pretože M'' má menej stavov ako M', existujú reťazce u, v, ktorých prijatím zo stavu q_0' prejdeme do dvoch rôznych stavov, zatiaľ čo ich prijatím zo stavu q_0'' prejdeme do toho istého stavu, teda $(q_0', u) \vdash^*_{M'} (q_1', e)$, $(q_0', v) \vdash^*_{M'} (q_2' e)$ a $q_1' \neq q_2'$, $(q_0'', u) \vdash^*_{M''} (q'', e)$, $(q_0'', v) \vdash^*_{M''} (q'', e)$. Preto reťazce u, v privedú automat M do rôznych stavov, povedzme p, r. To znamená, že existuje taký reťazec z, že práve jeden z reťazcov uz alebo vz patrí do $L(M)$. Potom ale reťazce uz, vz musia priviesť automat M'' do jedného stavu, povedzme s, pre ktorý platí $(q'', z) \vdash^*_{M''} (s, e)$. To ale znamená, že jeden z reťazcov uz, vz nemôže patriť do $L(M'')$, čo je v rozpore s predpokladom, že $L(M'') = L(M)$.

3.2 VZÁJOMNÝ VZŤAH KONEČNÝCH AUTOMATOV A REGULÁRNYCH GRAMATÍK

V tomto článku budeme skúmať vzťah medzi jazykmi, generovanými regulárnymi gramatikami a množinami, prijímanými konečnými automatmi. Keďže deterministické i nedeterministické konečné automaty rozpoznávajú tú istú triedu jazykov, nebudeme medzi nimi z tohto hľadiska robiť rozdiel.

Veta 3.3. Nech $G = (N, T, P, S)$ je regulárna gramatika. Potom existuje konečný automat $M = (Q, T, \delta, q_0, F)$ taký, že $L(M) = L(G)$.

Dôkaz. Dôkaz budeme aj teraz robiť pomocou konštrukcie konečného automatu pre danú regulárnu gramatiku, pričom konštrukciu budeme robiť tak, aby vytvorený automat prijímal práve množinu viet jazyka, generovaného príslušnou regulárnou gramatikou. Hľadaný konečný automat M pre gramatiku $G = (N, T, P, S)$ budeme konštruovať takto:

Nech $M = (Q, T, \delta, q_0, F)$. Za množinu stavov Q zoberieme množinu neterminálnych symbolov N z gramatiky G a pridáme k nej nový symbol A, ktorý nie je v N. Teda $Q = N \cup \{A\}$. Vstupnú abecedu stotožníme s množinou terminálnych symbolov (je už priamo v zápise). Za začiatočný stav zoberieme začiatočný symbol gramatiky G, teda $q_0 = S$. Množina koncových stavov F bude

$F = \{S, A\}$, ak $S \to e \in P$
$F = \{A\}$ v ostatných prípadoch

Zobrazenie δ definujeme takto: pre všetky $a \in T$, $B \in N$, $C \in N$ platí

$A \in \delta(B, a)$, ak $B \to a \in P$
$C \in \delta(B, a)$, ak $B \to aC \in P$
$\emptyset = \delta(B, a)$ v ostatných prípadoch

Dokážme teraz, že $L(M) = L(G)$.

1. Najprv dokážeme, že $L(G) \subseteq L(M)$, to znamená, že $w \in L(G) \to w \in L(M)$. Nech $w = a_1a_2 \dots a_n \in L(G)$ a $n \geqq 1$. V gramatike G potom existuje derivácia $S \Rightarrow a_1B_1 \Rightarrow a_1a_2B_2 \Rightarrow a_1a_2 \dots a_{n-1}B_{n-1} \Rightarrow a_1a_2 \dots a_n$, kde $B_1, B_2, \dots, B_{n-1}$ je nejaká postupnosť neterminálnych symbolov z N. Uvedená derivácia je podmienená existenciou prepisovacích pravidiel

$$\begin{aligned} &S_1 \to a_1B_1 \\ &B_1 \to a_2B_2 \\ &\vdots \\ &B_{n-2} \to a_{n-1}B_{n-1} \\ &B_{n-1} \to a_n \end{aligned}$$

Pre uvedené prepisovacie pravidlá však existujú zodpovedajúce prechody v konštruovanom automate, a to:

$$\begin{aligned} &B_1 \in \delta(S, a_1) \\ &B_2 \in \delta(B_1, a_2) \\ &\vdots \\ &B_{n-1} \in \delta(B_{n-2}, a_{n-1}) \\ &A \in \delta(B_{n-1}, a_n) \end{aligned}$$

Potom však možno vytvoriť postupnosť prechodov

$$(S, a_1a_2 \dots a_n) \vdash (B_1, a_2a_3 \dots a_n) \vdash \dots \vdash (B_{n-2}, a_{n-1}a_n) \vdash (B_{n-1}, a_n) \vdash (A, e)$$

To ale znamená, že platí

$$(S, a_1a_2 \dots a_n) \vdash^* (A, e)$$

a $A \in F$, teda reťazec $a_1a_2 \dots a_n$ je konštruovaným automatom prijatý. Keďže každé $w \in L(G)$ je z $L(M)$, platí

$$L(G) \subseteq L(M)$$

2. Teraz ukážeme, že platí $L(M) \subseteq L(G)$. Nech $w = a_1a_2 \dots a_n \in L(M)$ pre $n \geqq 1$. Potom ale automat M musí w prijať a platí $(S, w) \vdash^* (q_f, e)$ pre nejaké $q_f \in F$. Musí teda existovať postupnosť prechodov

$$(S, a_1a_2 \dots a_n) \vdash (B_1, a_2a_3 \dots a_n) \vdash \dots \vdash (B_{n-2}, a_{n-1}a_n) \vdash (B_{n-1}, a_n) \vdash (A, e)$$

pre nejakú postupnosť stavov $B_1, B_2, \dots, B_{n-1}$. Z konštrukcie konečného automatu M však teraz musí platiť

$$\begin{aligned} &B_1 \in \delta(S, a_1) \\ &B_2 \in \delta(B_1, a_2) \\ &\vdots \\ &B_{n-1} \in \delta(B_{n-2}, a_{n-1}) \\ &A \in \delta(B_{n-1}, a_n) \end{aligned}$$

Z toho však vyplýva, že v G musia existovať prepisovacie pravidlá

$$\begin{aligned} &S \to a_1B_1 \\ &B_1 \to a_2B_2 \\ &\vdots \\ &B_{n-2} \to a_{n-1}B_{n-1} \\ &B_{n-1} \to a_n \end{aligned}$$

a existuje teda aj derivácia

$$S \Rightarrow a_1B_1 \Rightarrow a_1a_2B_2 \Rightarrow \dots \Rightarrow a_1a_2 \dots a_{n-1}B_{n-1} \Rightarrow a_1a_2 \dots a_n$$

Keďže platí $S \Rightarrow^* w$, platí aj $w \in L(G)$. Každé $w \in L(M)$ je z $L(G)$, a teda $L(M) \subseteq L(G)$.

Uvedené podmienky neriešia eventuálny výskyt prázdneho slova v jazyku. Ak však $e \in L(G)$, tak $e \in L(M)$, pretože podľa konštrukcie M platí $S \in F$. Ak $e \in L(M)$, platí $S \in F$ a existuje prepisovacie pravidlo $S \to e$ v P, a preto aj $e \in L(G)$. Možno teda písať $L(M) = L(G)$.

Rozoberieme teraz obrátený prípad. Ku konečnému automatu M sa pokúsime vytvoriť regulárnu gramatiku G takú, že $L(G) = L(M)$. Poznamenávame

však, že z hľadiska aplikácie v prekladačoch je táto konštrukcia menej užitočná než konštrukcia automatu z gramatiky.

Veta 3.4. Nech $M = (Q, T, \delta, q_0, F)$ je konečný automat. Potom existuje taká regulárna gramatika $G = (N, T, P, S)$, že $L(G) = L(M)$.

Dôkaz. Pre jednoduchosť, ale bez straty na všeobecnosti, predpokladajme, že konečný automat M je deterministický. Gramatiku G vytvoríme nasledujúcim spôsobom:

$N = Q$

T bude zhodné s T z automatu

$S = q_0$

P: $B \to aC \in P$, ak $\delta(B, a) = C$
$B \to a \in P$, ak $\delta(B, a) = K$ a $K \in F$
$S \to e \in P$, ak $q_0 \in F$

Teraz je potrebné dokázať, že vytvorená gramtika skutočne generuje jazyk, ktorý prijíma automat M. Dôkaz je analogický ako vo vete 3.3, preto ho nebudeme robiť.

Príklad 3.8. Majme nasledujúcu regulárnu gramatiku:

$G = (\{C, D\}, \{+, -, 0, 1, ..., 9\}, P, C)$

kde

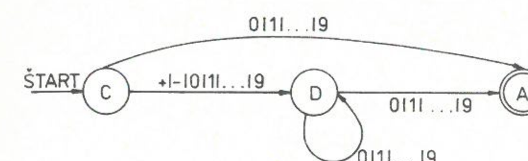
P: $C \to +D \mid -D \mid 0 \mid 1 \mid ... \mid 9 \mid 0D \mid ... \mid 9D$
$D \to 0 \mid 1 \mid ... \mid 9 \mid 0D \mid 1D \mid ... \mid 9D$

Gramatika G generuje celé čísla so znamienkom s prípustnými vedúcimi nulami. Skonštruujeme teraz ku G konečný automat M, ktorý bude prijímať rovnakú množinu slov ako G, teda množinu celých čísel s prípustnými nevýznamnými nulami vľavo.

Najprv formálne zapíšeme konečný automat v tvare $M = (Q, T, \delta, q_0, F)$ a vytvoríme jednotlivé objekty

$Q = N \cup \{A\} = \{A, C, D\}$
T zostáva z gramatiky
$q_0 = C$
$F = \{A\}$
δ: $\delta(D, x) = \{A\}$, kde $x = 0, 1, 2, ..., 9$ — pre pravidlá $D \to x$
$\delta(D, x) = \{D\}$, kde $x = 0, 1, 2, ..., 9$ — pre pravidlá $D \to xD$
Preto $\delta(D, x) = \{A, D\}$, kde $x = 0, 1, ..., 9$ a pravidlá $D \to x \mid xD$
$\delta(C, x) = \{D\}$, kde $x = +, -, 0, 1, ..., 9$ a pravidlo $C \to xD$
$\delta(C, x) = \{A\}$, kde $x = 0, 1, ..., 9$ a pravidlo $C \to x$
Preto $\delta(C, x) = \{A, D\}$, kde $x = 0, 1, ..., 9$ a pravidlá $C \to x \mid xD$

Možno vidieť, že automat je nedeterministický a zodpovedá mu prechodový diagram z *obr. 3.6*.



Obr. 3.6. Prechodový diagram pre automat z príkladu 3.8

Keďže automaty rozpoznávajú vety jazyka, stávajú sa prirodzenými modelmi analyzátorov prekladačov (lexikálneho i syntaktického). Na základe vety 3.3 možno takýto analyzátor, ktorý bude robiť lexikálnu analýzu, konštruovať automaticky z opisu regulárnej gramatiky. Na tom je založená aj automatizácia tvorby vybraných častí prekladača. To je tiež dôvod, prečo je konštrukcia z vety 3.3 pre prekladače užitočnejšia než konštrukcia z vety 3.4.

3.3 ZÁKLADNÉ VLASTNOSTI KONEČNÝCH AUTOMATOV A REGULÁRNYCH JAZYKOV

V tomto článku sa budeme zaoberať niektorými rozhodnuteľnými problémami z oblasti konečných automatov a regulárnych jazykov, problémami, na riešenie ktorých existujú zodpovedajúce algoritmy.

Veta 3.5. Množina slov prijatých konečným automatom s n stavmi je:

a) neprázdna práve vtedy, ak konečný automat prijme reťazec s dĺžkou menšou ako n,

b) nekonečná práve vtedy, ak konečný automat prijme reťazec s dĺžkou d, kde $n \leqq d < 2n$.

Dôkaz. a) Postačujúca podmienka je zrejmá. Ak konečný automat prijme reťazec s dĺžkou d, $d < n$, tak množina reťazcov prijatých konečným automatom je neprázdna.

Predpokladajme, že konečný automat $M = (Q, T, \delta, q_0, F)$ s n stavmi prijme nejaký reťazec. Nech w je najkratší prijatý reťazec. Predpokladajme, že $|w| \geqq n$, lebo v opačnom prípade nemáme čo dokazovať, keďže podmienky sú splnené. Pretože automat M má n stavov, pri prijatí w musí prejsť cez niektorý stav dvakrát. To znamená, že v Q môžeme nájsť taký stav q, že reťazec w sa dá napísať v tvare $w = w_1w_2w_3$, kde $w_2 \neq e$, pričom platí:

$(q_0, w_1) \vdash^* (q, e)$
$(q, w_2) \vdash^* (q, e)$
$(q, w_3) \vdash^* (q_f, e)$, kde $q_f \in F$

Potom ale w_1w_3 patrí do $L(M)$, pretože

$$(q_0, w_1w_3) \vdash^* (q, w_3) \vdash^* (q_f, e)$$

No vzhľadom na podmienku $w_2 \neq e$ platí $|w_1w_3| < |w|$, čo je v spore s predpokladom, že w je najkratší reťazec $L(M)$.

b) Najprv dokážeme, že ak $w \in L(M)$ a $n \leqq |w| < 2n$, tak jazyk $L(M)$ je nekonečný. Ak $w \in L(M)$ a $n \leqq |w| < 2n$, tak podobne ako v 1. časti dôkazu možno reťazec w písať v tvare $w = w_1w_2w_3$, kde $w_2 \neq e$, pričom platí :

$$(q_0, w_1) \vdash^* (q, e)$$
$$(q, w_2) \vdash^* (q, e)$$
$$(q, w_3) \vdash^* (q_f, e), \text{ kde } q_f \in F$$

Potom však platí aj

$$(q, w_2^i) \vdash^* (q, w_2^{i-1}) \vdash^* \ldots \vdash^* (q, w_2) \vdash (q, e)$$

pre $i \geqq 0$, takže do $L(M)$ patria všetky reťazce $w_1w_2^iw_3$ pre $i \geqq 0$ a jazyk $L(M)$ je nekonečný.

Predpokladajme teraz, že M rozpoznáva nekonečne veľa reťazcov w a žiadny $w \in L(M)$ nemá dĺžku s vlastnosťou $n \leqq |w| < 2n$. Zoberme najkratší reťazec w, ktorého dĺžka je väčšia alebo sa rovná $2n$. Keďže $|w| \geqq 2n$, možno reťazec w písať v tvare $w = w_1w_2w_3$, kde $1 \leqq |w_2| \leqq n$ a $w_1w_3 \in L(M)$. Pre dĺžku reťazca w_1w_3 platí, že alebo $n \leqq |w_1w_3| < 2n$, čo je spor s predpokladom, že žiadna veta jazyka nemá dĺžku s uvedenou vlastnosťou, alebo $|w_1w_3| \geqq 2n$, čo je spor s predpokladom, že w je najkratší reťazec s dĺžkou $|w| \geqq 2n$.

Veta 3.5 hovorí, že existujú algoritmy, ktorými možno rozhodnúť problém, či regulárny jazyk je prázdny, ako aj problém jeho konečnosti alebo nekonečnosti.

Nasledujúca veta nám umožní dokázať, že niektoré jazyky nie sú regulárne.

Veta. 3.6. Nech L je regulárny jazyk. Potom existuje taká konštanta p, že každý reťazec $w \in L$, taký že $|w| \geqq p$, možno písať v tvare $w = w_1w_2w_3$, kde $1 \leqq w_2 \leqq p$ a $w_1w_2i, w_3 \in L$ pre všetky $i \geqq 0$.

Dôkaz. Nech $M = (Q, T, \delta, q_0, F)$ je konečný automat s n stavmi a $L(M) = L$. Nech $p = n$. Ak $w \in L$ a $|w| \geqq n$, tak preskúšame postupnosť konfigurácií, ktorou prechádza automat M pri prijímaní reťazca w. Keďže v tejto postupnosti je najmenej $n + 1$ konfigurácií, musia sa medzi prvými $n + 1$ konfiguráciami nájsť také dve, ktoré majú rovnaký stav. Preto existuje taká postupnosť konfigurácií, pre ktorú platí

$$(q_0, w_1w_2w_3) \vdash^* (q_1, w_2w_3) \vdash^k (q_1, w_3) \vdash^* (q_2, e)$$

Pre nejaké q_1 a $0 < k \leqq n$. Teda platí $1 \leqq |w_2| \leqq n$. No potom môže automat pre ľubovoľné $i > 0$ urobiť nasledujúcu postupnosť prechodov:

$$\begin{aligned}(q_0\, w_1w_2^iw_3) &\vdash^* (q_1, w_2^iw_3)\\ &\vdash^+ (q_1, w_2^{i-1}w_3)\\ &\ \vdots\\ &\vdash^+ (q_1, w_2w_3)\\ &\vdash^+ (q_1, w_3)\\ &\vdash^* (q_2, e)\end{aligned}$$

Pretože reťazec $w = w_1w_2w_3$ patrí do jazyka L, patrí do L aj reťazec $w_1w_2^iw_3$ pre všetky $i > 0$. Analogicky možno urobiť dôkaz aj pre $i = 0$.

Príklad 3.9. Daný je jazyk $L = \{0^n1^n / n \geqq 1\}$. Treba zistiť, či jazyk L je regulárny. Pri zisťovaní tejto vlastnosti budeme vychádzať z vety 3.6.

Predpokladajme, že jazyk L je regulárny. Pre dostatočne veľké n možno potom reťazec 0^n1^n zapísať v tvare $w_1w_2w_3$, pričom $w_2 \neq e$ a $w_1w_2^iw_3 \in L$ pre všetky $i \geqq 0$. Pre zápis reťazca $w_1w_2w_3$ existujú nasledujúce možnosti, pri ktorých použijeme skrátený zápis a^n namiesto $\{a\}^n$.

$$0^k0^d1^n \quad \text{a} \quad k + d = n$$
$$0^n1^d1^k \quad \text{a} \quad k + d = n$$
$$0^k0^d1^r1^s \quad \text{a} \quad k + d = r + s$$

Znova však zdôrazníme požiadavku $w_1w_2w_3 \in L$. V našom prípade však ľahko zistíme, že

$$0^k(0^d)^i1^n \notin L$$
$$0^n(1^d)^i1^k \notin L$$
$$0^k(0^d1^r)^i1^s \notin L$$

pre ľubovoľné i okrem $i = 1$, a preto tento jazyk nie je regulárny.

Na záver ešte uvedieme bez dôkazu niektoré vlastnosti regulárnych jazykov.

1. Regulárne jazyky sú uzavreté vzhľadom na operáciu zjednotenia. To znamená, že ak sú jazyky L_1 a L_2 regulárne, aj jazyk $L = L_1 \cup L_2$ je regulárny.

2. Regulárne jazyky sú uzavreté vzhľadom na operáciu doplnku.

3. Trieda množín, ktoré sa dajú rozpoznať konečným automatom, tvorí Boolovu algebru.

Cvičenia

1. Implementujte programovú reprezentáciu konečného automatu z príkladu 3.4.

2. Dokážte vetu 3.4.

3. Vytvorte ekvivalentný konečný automat pre gramatiky, špecifikujúce
   a) identifikátor,
   b) vyhradené slová
   jazyka pascal.

4. Daný je nedeterministický konečný automat

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_0, \{q_2, q_4\})$,

kde δ je definované takto (*tab. 3.4*):

Tabuľka 3.4

δ	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$

Vytvorte taký deterministický konečný automat M', že $L(M') = L(M)$

5. Dokážte, že pre automat M vytvorený algoritmom 3.1 platí $L(M') = L(M)$.

Ďalším špecifikačným prostriedkom pre regulárne jazyky sú regulárne výrazy.

Definícia 3.10. *Regulárne výrazy* nad abecedou T, ktoré označujú, *regulárne jazyky*, nazývané v tejto súvislosti často regulárne množiny, sú definované takto:

a) $\emptyset$ je regulárny výraz, ktorý označuje regulárnu množinu $\emptyset$,
b) e je regulárny výraz, ktorý označuje regulárnu množinu $\{e\}$,
c) ak $t \in T$, tak t je regulárny výraz, ktorý označuje regulárnu množinu $\{t\}$,
d) ak a, b sú regulárne výrazy označujúce regulárne množiny A, B, tak
— $(a + b)$ je regulárny výraz označujúci $A \cup B$,
— (ab) je regulárny výraz označujúci AB,
— $(a)^*$ je regulárny výraz označujúci A^*,
e) regulárnym výrazom je iba výraz vytvorený na základe bodov a) až d).

6. Dokážte, že množiny $\emptyset$, $\{e\}$, $\{t\}$ pre všetky $t \in T$ sú vpravo lineárnymi jazykmi (jazyky generované gramtikami s vytvárajúcimi pravidlami $A \to w$ alebo $A \to wB$, kde A, $B \in N$, $w \in T^*$).

7. Dokážte, že ak L_1 a L_2 sú vpravo lineárne jazyky, tak vpravo lineárne sú aj jazyky $L_1 \cup L_2$, L_1L_2, L_1^*.

8. Dokážte, že jazyk je regulárnou množinou práve vtedy, ak je vpravo lineárny.

4 BEZKONTEXTOVÉ GRAMATIKY A JAZYKY

Regulárne gramatiky a jazyky majú celý rad dobrých vlastností, no ich praktické použitie v oblasti programovacích jazykov a prekladačov je veľmi obmedzené. Na špecifikáciu vyšších konštrukcií programovacích jazykov používame bezkontextové gramatiky a vystačíme s nimi pri špecifikácii väčšiny konštrukcií programovacích jazykov.

Pri špecifikácii programovacích jazykov musíme však mať na pamäti aj skutočnosť, že programy bude potrebné preložiť, že pre daný jazyk treba vytvoriť prekladač. Tu už vystupuje do popredia aj význam slov jazyka. Uvidíme, že existujú slová jazyka, ktorých význam nemusí byť jednoznačný. Z hľadiska prekladu to má ďalekosiahly význam, pretože sa môže uskutočniť iným spôsobom, než zamýšľal programátor. Okrem toho pri špecifikácii programovacích jazykov požadujeme takú špecifikáciu, ktorá umožňuje jednoduchý a rýchly preklad. Samozrejme, nie všetky bezkontextové gramatiky túto vlastnosť majú. Zaujíma nás preto, či nemožno gramatiku opisujúcu daný jazyk transformovať na ekvivalentú gramatiku s priaznivejšími vlastnosťami. Ukážeme, že takýchto transformácií existuje veľa, navyše existujú algoritmy, ktoré umožnia zistiť viaceré vlastnosti. Týmito algoritmami sa budeme tiež zaoberať.

Začneme najprv prostriedkami, ktoré umožnia zistiť niektoré vlastnosti bezkontextových gramatík.

4.1. DERIVAČNÉ STROMY — VLASTNOSTI BEKONTEXTOVÝCH GRAMATÍK

Viaceré vlastnosti bezkontextových gramatík závisia od derivácie, ktorou sa generuje dané slovo jazyka. Preto je vhodné, aby sme na opis derivácie použili taký formalizmus, ktorý nám o derivácii poskytne čo najviac informácií. Takým je derivačný strom.

Najprv uvedieme neformálnu definíciu stromu. Strom je konečná množina

vrcholov alebo uzlov, spojených orientovanými hranami, ktoré spĺňajú nasledujúce tri podmienky:

1. Existuje práve jeden vrchol, do ktorého nevstupuje žiadna hrana. Tento vrchol nazývame *koreň stromu.*

2. Ku každému vrcholu existuje postupnosť orientovaných hrán z koreňa stromu do daného vrcholu. Strom je teda *súvislý.*

3. Do každého vrcholu s výnimkou koreňa stromu vstupuje *práve jedna hrana.* V strome neexistujú cykly.

Definícia derivačného stromu vyžaduje, aby sme ešte objasnili niektoré pojmy. Pojmy vstupuje a vystupuje znamenajú: Ak vrcholy u, v sú spojené orientovanou hranou z u do v hrana *vystupuje* z u a *vstupuje* do v. Vrchol u nazývame *priamy predchodca* vrcholu v, vrchol v *priamy nasledovník* vrcholu u. Vrcholy, ktoré nemajú nasledovníkov, nazývame *koncové.* Vrcholy derivačného stromu budeme spájať s *terminálnymi* alebo *neterminálnymi symbolmi,* prípadne *prázdnymi reťazcami.* O takomto strome budeme potom hovoriť, že je *ohodnotený.*

Ak má vrchol stromu viac nasledovníkov, je v niektorých prípadoch výhodné ich usporiadať. My budeme používať usporiadanie definované pomocou *ohodnotenia* alebo jednoducho definované výskytom vrcholu v strome zľava doprava.

Definícia 4.1. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Orientovaný ohodnotený a usporiadaný strom D je potom *derivačným stromom* pre G, ak spĺňa nasledujúce podmienky:

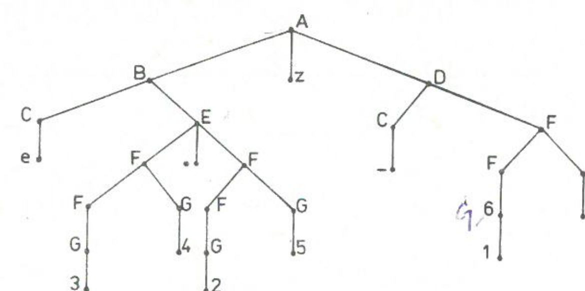
1. Každý vrchol je ohodnotený symbolom z $N \cup T \cup \{e\}$.
2. Ohodnotenie koreňa stromu je S.
3. Ak vrchol má aspoň jedného nasledovníka, je ohodnotený symbolom z N.
4. Ak $u_1, u_2, \ldots, u_k$, sú priami nasledovníci vrcholu u, ktorý je ohodnotený symbolom $A \in N$, a nasledovníci sú ohodnotení zľava doprava $A_1, A_2, \ldots, A_k$, tak v P musí existovať prepisovacie pravidlo $A \to A_1A_2 \ldots A_k$.

Príklad 4.1. Daná je gramatika

$G = (\{A, B, C, D, E, F, G\}, \{+, -, ., 0, 1, \ldots, 9, z\}, P, A)$

P: $A \to B \mid BzD$
$B \to CE$
$C \to + \mid - \mid e$
$D \to CF$
$E \to F. \mid F.F \mid .F$
$F \to G \mid FG$
$G \to 0 \mid 1 \mid 2 \mid \ldots \mid 9$

Potom možno vytvoriť nasledujúci derivačný strom (*obr. 4.0*). O tom, že ide skutočne o derivačný strom pre G, sa možno ľahko presvedčiť. Koreň je skutočne ohodnotený začiatočným symbolom A gramatiky G. Má troch nasledovníkov, ktorí sú ohodnotení postupne zľava doprava B, z, D. Vidíme, že v P existuje



Obr. 4.0 Derivačný strom k príkladu 4.1

prepisovacie pravidlo $A \to BzD$. Ľahko sa tiež možno presvedčiť, že každý vrchol ohodnotený neterminálnym symbolom má nejakých priamych nasledovníkov a že zreťazenie ich ohodnotení tvorí pravú stranu prepisovacieho pravidla zodpovedajúceho ohodnoteniu ich predchodcu. Rovnako možno vidieť, že žiadny vrchol ohodnotený terminálnym symbolom nemá nasledovníkov, a preto je koncovým vrcholom derivačného stromu.

Definícia 4.2. *Rezom stromu D* nazývame takú množinu vrcholov C stromu D, ktorá spĺňa tieto podmienky:

1. Žiadne dva vrcholy z C neležia na jednej ceste v D.
2. K C nemožno pridať žiaden vrchol z D, aby sa neporušila podmienka 1.

Rez stromu má významné postavenie v derivačnom strome, ktorý je usporiadaný a ohodnotený. Zreťazením ohodnotení rezu zľava doprava dostávame reťazec z $(N \cup T)^*$, zreťazením koncových vrcholov reťazec z T^*. Uvidíme, že tieto reťazce sú vetnými formami danej gramatiky.

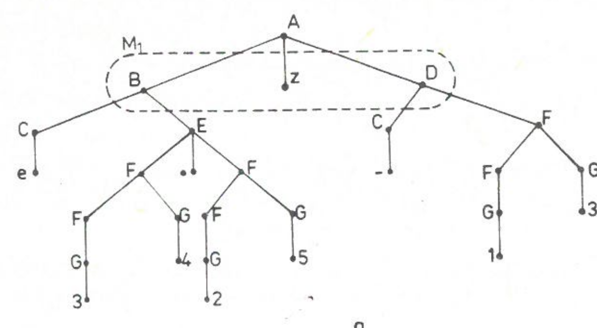
Definícia 4.3. *Čelom derivačného stromu* nazývame reťazec, ktorý dostaneme zreťazením ohodnotení koncových vrcholov zľava doprava. *Čelom rezu stromu D* nazývame reťazec, ktorý dostaneme zreťazením ohodnotení rezu stromu D zľava doprava.

Podstromom derivačného stromu budeme nazývať štruktúru, ktorá sa skladá z nejakého vrcholu derivačného stromu spolu so všetkými jeho nasledovníkmi i s ohodnoteniami a hranami, ktoré ich spájajú.

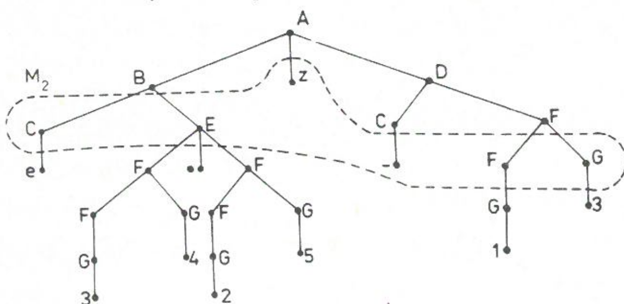
Príklad 4.2. Majme derivačný strom z *obr. 4.1.* Rezmi sú potom napr. množiny

$M_1 = \{B, z, D\}$
$M_2 = \{C, E, z, -, F, G\}$
$M_3 = \{C, 3, 4, ., 2, 5, z, -, 1, 3\}$

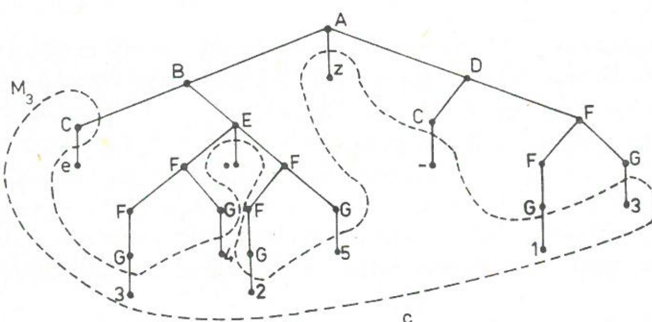
zobrazené na *obr. 4.1. a, b, c.*



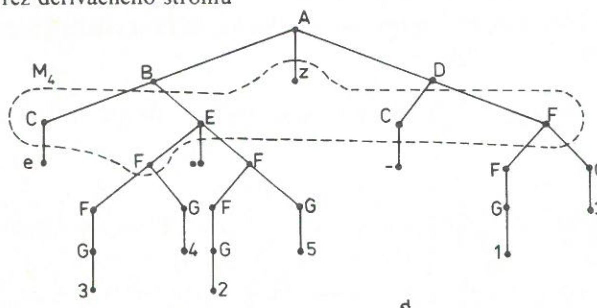
Obr. 4.1. Derivačný strom k príkladu 4.1. a) rez derivačného stromu



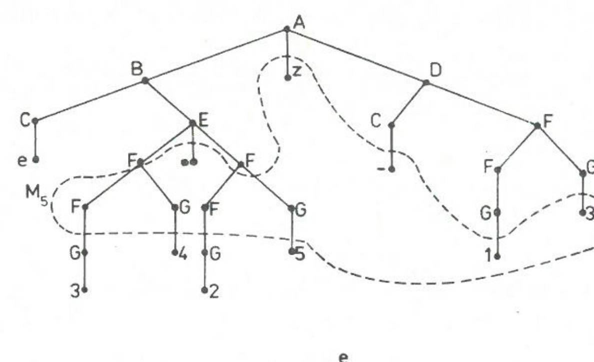
b) rez derivačného stromu



c) rez derivačného stromu



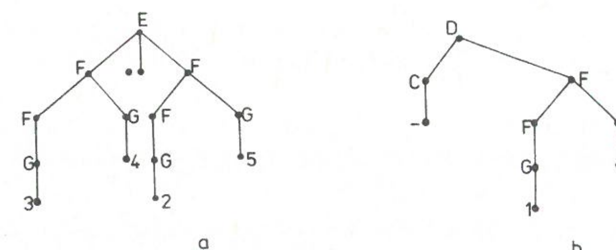
d) množina vrcholov, ktorá nie je rezom daného derivačného stromu



e) množina vrcholov, ktorá nie je rezom daného derivačného stromu

Rezmi nie sú napr. množiny M_4 a M_5, zobrazené na *obr. 4.1 d, e.*

$M_4 = \{C, F, E, z, C, F\}$, pretože E, F sú na jednej ceste z A.
$M_5 = \{F, G, ., F, G, z, -, 1, 3\}$, možno pridať C.
Podstromami sú (*obr. 4.2a, b*).



Obr. 4.2. a) podstrom derivačného stromu z príkladu 4.1, b) podstrom derivačného stromu z príkladu 4.1

Čelom derivačného stromu D z príkladu 4.1 je reťazec 34.25z-13.
Čelom podstromov D_1 a D_2 sú: 34.25, – 13.
Čelom rezov M_1, M_2 a M_3 sú:

M_1: *BzD*
M_2: *CEz-FG*
M_3: *C*34.25*z*-13

Ak sa na uvedené podstromy pozrieme z hľadiska gramatiky alebo derivačných stromov, tak vidíme, že znova ide o určité derivačné stromy zodpovedajúce nejakej gramatike. Ide o určitú „podgramatiku" pôvodnej gramatiky s novým začiatočným symbolom, ktorým je ohodnotený koreň daného podstromu.

Z hľadiska derivácie zodpovedá podstrom určitej čiastočnej derivácii v pôvodnej gramatike.

Lema 4.1. Nech $G = (N, T, P, S)$ je bezkontextová gramatika a $\alpha_0, \alpha_1, \ldots, \alpha_n$ je derivácia α_n z $\alpha_0 = S$. Potom možno v G zostrojiť derivačný strom D, v ktorom je α_n čelo a $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ sú čelá rezov.

Dôkaz. Vytvoríme takú postupnosť derivačných stromov D_i pre $0 \leqq i \leqq n$, v ktorých α_i je čelo stromu D_i. Nech D_0 je strom pozostávajúci z jediného vrcholu ohodnoteného S. Pripusťme, že $\alpha_i = \beta_i A \gamma_j$ a po aplikovaní prepisovacieho pravidla $A \to X_1 X_2 \ldots X_k$ dostaneme $\alpha_{i+1} = \beta_i X_1 X_2 \ldots X_k \gamma_j$. Strom D_{i+1} potom dostaneme zo stromu D_i tak, že ku koncovému vrcholu ohodnotenému symbolom A (ide o $(|\beta_i| + 1)$-vý symbol čela stromu D_i) pridáme k priamych nasledovníkov, ktorých zľava doprava ohodnotíme $X_1, X_2, \ldots, X_k$. Je jasné, že čelom stromu D_{i+1} bude práve α_{i+1}. Strom D_n bude hľadaným derivačným stromom D.

Lema 4.2. Nech D je derivačný strom bezkontextovej gramatiky $G = (N, T, P, S)$, ktorý má čelo α. Potom existuje derivácia α v G, čiže $S \Rightarrow^* \alpha$.

Dôkaz. Nech $C_0, C_1, \ldots, C_n$ je taká postupnosť rezov stromu D, ktoré spĺňajú nasledujúce podmienky:

1. C_0 obsahuje iba koreň stromu D.
2. C_{i+1} pre $0 \leqq i < n$ sa získa z C_i náhradou jedného neterminálneho symbolu ohodnoteniami priamych nasledovníkov daného vrcholu.
3. C_n je čelo stromu D.

Je jasné, že aspoň jedna takáto postupnosť existuje. Ak α_i je čelo rezu C_i, tak $\alpha_0, \alpha_1, \ldots, \alpha_n$ je derivácia reťazca α_n z α_0 v G. Keďže C_0 je spojené s koreňom stromu, platí $S \Rightarrow^* \alpha_n$.

Veta 4.1. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Derivácia $S \Rightarrow^* \alpha$ potom existuje vtedy a len vtedy, ak pre G existuje derivačný strom s čelom α.

Dôkaz. Dôkaz vyplýva priamo z liem 4.1 a 4.2.

P r í k l a d 4.3. Majme znova gramatiku

$G = (\{A, B, C, D, E, F, G\}, \{+, -, ., z, 0, 1, \ldots, 9\}, P, A,)$

P: $A \to B \mid BzD$
$B \to CE$
$C \to + \mid - \mid e$
$D \to CF$
$E \to F. \mid F.F \mid .F$
$F \to G \mid FG$
$G \to 0 \mid 1 \mid \ldots \mid 9$

Reťazec 34.25z-13 potom možno získať napr. nasledujúcimi deriváciami:

1. $A \Rightarrow BzD \Rightarrow CEzD \Rightarrow EzD \Rightarrow F.FzD \Rightarrow FG.FzD \Rightarrow GG.FzD \Rightarrow$
$\Rightarrow 3G.FzD \Rightarrow 34.FzD \Rightarrow 34.FGzD \Rightarrow 34.GGzD \Rightarrow 34.2GzD \Rightarrow$
$\Rightarrow 34.25zD \Rightarrow 34.25zCF \Rightarrow 34.25z\text{-}F \Rightarrow 34.25z\text{-}FG \Rightarrow 34.25z\text{-}GG \Rightarrow$
$\Rightarrow 34.25z\text{-}1G \Rightarrow 34.25z\text{-}13.$

2. $A \Rightarrow BzD \Rightarrow BzCF \Rightarrow BzCFG \Rightarrow BzCF3 \Rightarrow BzCG3 \Rightarrow BzC13 \Rightarrow$
$\Rightarrow Bz\text{-}13 \Rightarrow CEz\text{-}13 \Rightarrow CF.Fz\text{-}13 \Rightarrow CF.FGz\text{-}13 \Rightarrow CF.F5z\text{-}13 \Rightarrow$
$\Rightarrow CF.G5z\text{-}13 \Rightarrow CF.25z\text{-}13 \Rightarrow CFG.25z\text{-}13 \Rightarrow CF4.25z\text{-}13 \Rightarrow$
$\Rightarrow CG4.25z\text{-}13 \Rightarrow C34.25z\text{-}13 \Rightarrow 34.25z\text{-}13.$

Ľahko možno vidieť, že počet derivácií slova 34.25z-13 je väčší, než sme uviedli v príklade 4.3. Spomedzi všetkých možných derivácií však tieto dve majú osobitné postavenie.

Definícia 4.4. Deriváciu $\alpha_0, \alpha_1, \ldots, \alpha_k$, v ktorej každú priamu deriváciu $\alpha_i \Rightarrow \alpha_{i+1}$ pre $0 \leqq i < k$ realizujeme tak, že vo vetnej forme α_i nahrádzame prvý neterminálny symbol zľava, nazývame *ľavá derivácia*; ak nahrádzame prvý symbol sprava, je to *pravá derivácia*. *Vetné formy* v ľavej (pravej) derivácii nazývame *ľavé* (*pravé*).

Ak sa vrátime k predchádzajúcemu príkladu, tak 1. derivácia je ľavá, 2. derivácia je pravá. Ľavá a pravá derivácia sú s derivačným stromom viazané nasledujúcim spôsobom: pre každý derivačný strom existuje iba jedna pravá a jedna ľavá derivácia.

Poznamenávame, že v literatúre sa možno stretnúť aj s názvami *najľavejšia* a *najpravejšia* derivácia. My budeme aj naďalej používať ľavú a pravú deriváciu s definovaným významom.

Rozoberieme ešte vzťah derivácie a zodpovedajúceho derivačného stromu z hľadiska jedinečnosti derivačného stromu k danej derivácii. Derivačný strom síce skrýva spôsob svojho vytvárania, na druhej strane však možno štruktúru derivačného stromu využiť na určenie štruktúry vety jazyka. Majme napr. reťazec $5 + 6 \times 3$, ktorý predstavuje aritmetický výraz. Ak nevieme nič viac, iba vykonávať zodpovedajúce aritmetické operácie, tak nie je jednoznačné, či najprv vykonať sčítanie a potom násobenie, teda vykonávať operácie napr. zľava doprava, alebo pri vykonávaní použiť nejaké iné pravidlo. Ak by sme chceli určiť niekomu, povedzme počítaču, postup, ako vyčísliť daný výraz, využili by sme našu znalosť o tom, že priorita násobenia je vyššia ako priorita sčítania, a preto vykonáme najprv násobenie. Pri programovacích jazykoch je takáto znalosť „zabudovaná" priamo v gramatike. Možno to vidieť z derivačného stromu v nasledujúcom príklade.

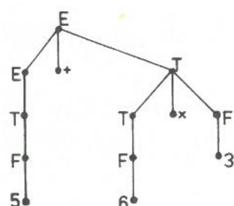
P r í k l a d 4.4. Daná je gramatika $G = (\{E, T, F\}, \{+, \times, 3, 5, 6\}, P, E)$

P: $E \to E + T \mid T$
$T \to T \times F \mid F$
$F \to 3 \mid 5 \mid 6$

Reťazec $5 + 6 \times 3$ možno potom získať deriváciou

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow 5 + T \Rightarrow 5 + T \times F \Rightarrow$
$\Rightarrow 5 + F \times F \Rightarrow 5 + 6 \times F \Rightarrow 5 + 6 \times 3$

Zo štruktúry derivačného stromu k reťazcu $5 + 6 \times 3$ vidieť (*obr. 4.3*), že 6, 3 sú operandy patriace k operátoru × (krát). Vzniká však otázka, či k danému reťazcu existuje iba jeden derivačný strom, či neexistuje iný derivačný strom, ktorý by umožňoval inú interpretáciu reťazca $5 + 6 \times 3$. Aj keď v tomto prípade taký derivačný strom neexistuje, existujú gramatiky, v ktorých k jednému reťazcu možno vytvoriť viac derivačných stromov, líšiacich sa navzájom svojou štruktúrou. Ukážeme to v nasledujúcom príklade.



Obr. 4.3. Derivačný strom k príkladu 4.4

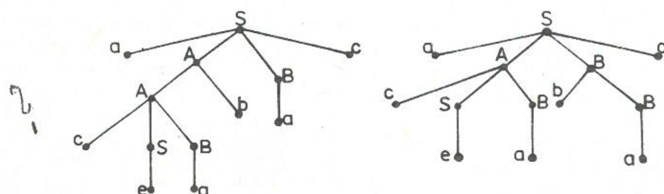
Príklad 4.5. Majme gramatiku G so začiatočným symbolom S a prepisovacími pravidlami P:

$S \to aABc \mid e$
$A \to cSB \mid Ab$
$B \to bB \mid a$

Pre reťazec *acabac* možno potom vyvoriť ľavé a pravé derivácie:

1. Ľavé a) $S \Rightarrow aABc \Rightarrow aAbBc \Rightarrow acSBbBc \Rightarrow acBbBc \Rightarrow acabBc \Rightarrow acabac$,
   b) $S \Rightarrow aABc \Rightarrow acSBBc \Rightarrow acBBc \Rightarrow acaBc \Rightarrow acabBc \Rightarrow \Rightarrow acabac$.
2. Pravé: a) $S \Rightarrow aABc \Rightarrow aAac \Rightarrow aAbac \Rightarrow acSBbac \Rightarrow acSabac \Rightarrow acabac$,
   b) $S \Rightarrow aABc \Rightarrow aAbBc \Rightarrow aAbac \Rightarrow acSBbac \Rightarrow acSabac \Rightarrow$ $\Rightarrow acabac$.

Ďalej možno vytvoriť dva derivačné stromy (pozri *obr. 4.4*).



Obr. 4.4. Derivačný strom k príkladu 4.5

Ako vidieť, v danej gramatike existujú pre ten istý reťazec dve rôzne ľavé derivácie, dve rôzne pravé derivácie, ako aj dva rôzne derivačné stromy.

Definícia 4.5. Bezkontextová gramatika $G = (N, T, P, S)$ je *nejednoznačná* (*viacznačná*), ak existuje aspoň jedno také slovo $w \in L(G)$, že pre w existuje viac vzájomne odlišných derivačných stromov. Ekvivalentne možno povedať, že gramatika G je nejednoznačná, ak existuje také slovo $w \in L(G)$, že pre w existuje viac ľavých (pravých) derivácií.

Uvedieme ešte jeden príklad nejednoznačnej gramatiky, ktorý je známy z programovacích jazykov.

Príklad 4.6. Daná je gramatika $G = (\{S\}, \{a, b,$ **if**, **then**, **else**$\}, P, S)$

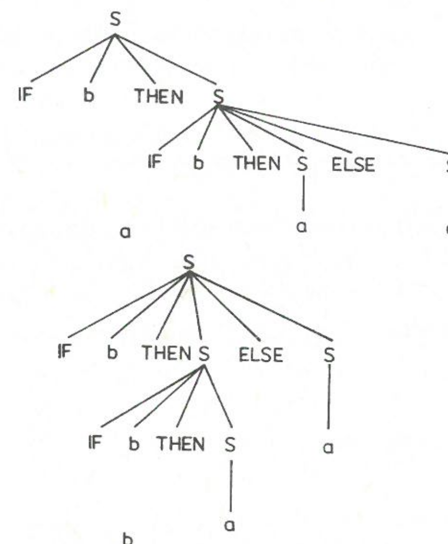
P: $S \to$ **if** b **then** S **else** S
$S \to$ **if** b **then** S
$S \to a$

Gramatika G je nejednoznačná, pretože pre slovo **if** b **then if** b **then** a **else** a existujú dve ľavé derivácie.

a) $S \Rightarrow$ **if** b **then** $S \Rightarrow$ **if** b **then if** b **then** S **else** $S \Rightarrow$ **if** b **then if** b **then** a **else** $S \Rightarrow$ $\Rightarrow$ **if** b **then if** b **then** a **else** a,

b) $S \Rightarrow$ **if** b **then** S **else** $S \Rightarrow$ **if** b **then if** b **then** S **else** $S \Rightarrow$ **if** b **then if** b **then** a **else** $S \Rightarrow$ **if** b **then if** b **then** a **else** a.

Príslušné derivačné stromy sú na *obr. 4.5a, b*.



Obr. 4.5. a) derivačný strom ku gramatike z príkladu 4.6 a k reťazcu **if** b **then if** b **then** a **else** a, b) derivačný strom ku gramatike z príkladu 4.6 a k reťazcu **if** b **then if** b **then** a **else** a

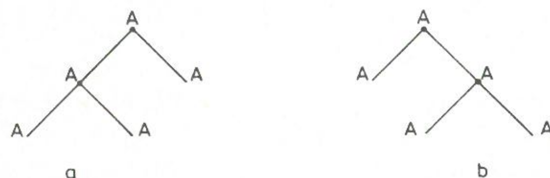
Treba ešte zdôrazniť, že nejednoznačnosť sa týka gramatiky, a nie jazyka. V mnohých prípadoch možno vytvoriť gramatiku, ktorá generuje daný jazyk a je jednoznačná. Existujú však jazyky, ktoré jednoznačnou gramatikou generovať nemožno. Takéto jazyky nazývame *vnútorne* (*inherentne*) *nejednoznačné*.

Otázka týkajúca sa jednoznačnosti bezkontextových gramatík je nerozhodnuteľná. Neexistuje algoritmus, ktorý by rozhodol či ľubovoľná bezkontextová gramatika je alebo nie je jednoznačná. Existujú však gramatiky, ktorých nejednoznačnosť možno zistiť, dokonca ich možno transformovať na ekvivalentnú gramatiku, ktorá je jednoznačná. Napríklad gramatiku z príkladu 4.6 možno transformovať na ekvivalentnú gramatiku

$G_1 = (\{S_1, S_2\}, \{a, b, \textbf{if}, \textbf{then}, \textbf{else}\}, P, S_1)$
P: $S_1 \to \textbf{if}\ b\ \textbf{then}\ S_1 \mid \textbf{if}\ b\ \textbf{then}\ S_2\ \textbf{else}\ S_1 \mid a$
$S_2 \to \textbf{if}\ b\ \textbf{then}\ S_2\ \textbf{else}\ S_2 \mid a$

ktorá je jednoznačná.

Nejednoznačnosť niektorých bezkontextových gramatík možno rozpoznať z prepisovacích pravidiel. Napríklad gramatika, ktorá obsahuje prepisovacie pravidlá tvaru $A \to AA \mid a$, je nejednoznačná. Možno to ľahko zistiť z toho, že podreťazec AAA pripúšťa dve rôzne derivácie s derivačnými stromami, ako možno vidieť na *obr. 4.6a, b*.



Obr. 4.6. a), b) derivačné stromy pre gramatiku s prepisovacími pravidlami $A \to AA \mid a$

Vzhľadom na špeciálny tvar prepisovacích pravidiel vedúcich k nejednoznačnosti gramatiky s týmito pravidlami možno danú gramatiku transformovať modifikovaním prepisovacích pravidiel $A \to AA \mid a$ a ich nahradením jednou z nasledujúcich dvojíc prepisovacích pravidiel:

$A \to AB \mid B$
$B \to a$

alebo

$A \to BA \mid B$
$B \to a$

Podobným príkladom prepisovacieho pravidla spôsobujúcim nejednoznačnosť

je prepisovacie pravidlo $A \to A\ \alpha\ A$. Takýto prípad je napr. gramatika

$G = (\{E\}, \{+, \times, a, (,)\}, P, E)$, kde P: $E \to E + E \mid E \times E \mid (E) \mid a$

Aj dvojica pripisovacích pravidiel $A \to \alpha A \mid A\beta$ vedie k nejednoznačnosti, pretože reťazec $\alpha A \beta$ má dve ľavé derivácie, a to:

$A \Rightarrow \alpha A \Rightarrow \alpha A \beta$
$A \Rightarrow A \beta \Rightarrow \alpha A \beta$

4.2 TRANSFOMÁCIE BEZKONTEXTOVÝCH GRAMATÍK

Už v predchádzajúcich kapitolách sme ukázali, že daný jazyk možno generovať niekoľkými rôznymi gramatikami. V skutočnosti, ak $L = L(G)$ je jazyk generovaný gramatikou G, existuje nekonečne veľa gramatík, ktoré sú ekvivalentné s gramatikou G, t. j. všetky generujú jazyk L. Tieto gramatiky sa môžu líšiť v množinách neterminálnych alebo terminálnych symbolov, v prepisovacích pravidlách, ktoré špecifikujú syntaktickú štruktúru jazyka, alebo aj v začiatočných symboloch. Transformáciou bezkontextovej gramatiky potom rozumieme postup, ktorým z danej gramatiky získame ekvivalentnú gramatiku, pričom zvyčajne chceme, aby výsledná gramatika mala určitú požadovanú vlastnosť.

Vo všeobecnosti neexistuje algoritmická metóda, ktorá by umožňovala zostrojiť alebo modifikovať gramatiku jazyka tak, aby generovaný jazyk mal požadovanú syntaktickú štruktúru. Problém transformácie je komplikovaný aj tým, že je algoritmicky nerozhodnuteľné, či sú dve bezkontextové gramatiky ekvivalentné.

Existuje však celý rad užitočných transformácií gramatík, ktoré dovoľujú modifikovať gramatiku bez toho, že by sa zmenil generovaný jazyk. Takými transformáciami sa budeme teraz zaoberať.

4.2.1 Odstránenie nadbytočných symbolov gramatiky

Formálna definícia gramatiky pripúšťa, aby množiny N, T, P obsahovali prvky, ktoré sa nemôžu uplatniť v derivácii žiadnej vety jazyka. Prítomnosť takýchto nadbytočných symbolov a pravidiel môže byť dôsledkom chyby v návrhu gramatiky alebo úpravy gramatiky a v každom prípade zbytočne zväčšuje rozsah gramatiky.

Definícia 4.6. Nech $G = (N, T, P, S)$ je gramatika. Symbol X z $N \cup T$ je v gramatike G *nadbytočný*, ak neexistuje v G derivácia

$S \overset{*}{\Rightarrow} xXz \overset{*}{\Rightarrow} xyz$

taká, že $x, y, z \in T^*$. Ak pre X neexistuje v G derivácia

$$S \overset{*}{\Rightarrow} \alpha X \beta;\ \alpha,\ \beta \in (N \cup T)^*$$

tak X sa nazýva *nedostupný symbol.*

Zrejme každý nedostupný symbol je nadbytočný. Nadbytočnými sú však aj neterminálne symboly, z ktorých nemožno v G derivovať reťazec terminálnych symbolov alebo prázdny reťazec.

Najskôr ukážeme algoritmus vyhľadávania tých neterminálnych symbolov, z ktorých možno generovať reťazce z T^*.

Definícia 4.7. Nech $G = (N, T, P, S)$ je gramatika. Označme N_T, $N_T \subseteq N$, množinu

$$N_T = \{A/A \overset{*}{\Rightarrow} w,\ w \in T^*\}$$

Algoritmus 4.1.

Výpočet množiny N_T.

Vstup: gramatika $G = (N, T, P, S)$

Výstup: množina $N_T = \{A/A \overset{*}{\Rightarrow} w,\ w \in T^*\}$.

Metóda: budeme vytvárať množiny $N_0, N_1, N_2, \ldots$ neterminálnych symbolov podľa nasledujúceho rekurentného postupu:

1. $N_0 := \emptyset$, $i := 1$
2. $N_i := \{A/A \to \alpha$ je pravidlo z P, $\alpha \in (N_{i-1} \cup T)^*\} \cup N_{i-1}$
3. Ak $N_i \neq N_{i-1}$, tak $i := i + 1$ a vráť sa na krok 2; inak $N_T := N_i$

Dôkaz. Algoritmus 4.1 vykoná krok 2 najviac $(|N| + 1)$-krát. Pretože N je konečná množina, je zaručená konečnosť výpočtu množiny N_T. Dokážeme teraz jeho správnosť:

$$A \in N_T \text{ vtedy a len vtedy, ak } A \overset{*}{\Rightarrow} w,\ w \in T^*$$

Dôkaz príslušných implikácií vykonáme matematickou indukciou:

1. $A \in N_i$ implikuje $A \overset{*}{\Rightarrow} w$, $w \in T^*$

Pre $i = 0$ táto implikácia platí, pretože $N_0 = \emptyset$. Predpokladajme, že implikácia platí pre i a nech $A \in N_{i+1}$. Ak $A \in N_i$, tak dokazované tvrdenie platí pre $i + 1$ triviálne. Nech teda $A \in N_{i+1} - N_i$. Potom však existuje, vzhľadom na definíciu N_{i+1}, pravidlo $A \to X_1X_2 \ldots X_k$, kde pre $j = 1, 2, \ldots, k$ platí alebo $X_j \in T$, alebo $X_j \in N_i$. Vzhľadom na indukčnú hypotézu teda platí $X_j \overset{*}{\Rightarrow} w_j$, $w_j \in T^*$ pre všetky j. Existuje teda derivácia $A \Rightarrow X_1X_2 \ldots X_k \overset{*}{\Rightarrow} w_1w_2 \ldots w_k = w$ a dokazovaná implikácia platí pre $i + 1$.

2. $A \overset{n}{\Rightarrow} w$, $w \in T^*$, implikuje $A \in N_i$

pre nejaké $i (N_i \subseteq N_T)$. Ak $n = 1$, tak $i = 1$, pretože $N_1 = \{A/A \to \alpha$ je v P a $\alpha \in T^*\}$. Predpokladajme, že dokazované tvrdenie platí pre n a nech $A \overset{n+1}{\Rightarrow} w$. Predpoklad $A \overset{n+1}{\Rightarrow} w$ môžeme prepísať na tvar $A \Rightarrow X_1X_2 \ldots X_k \overset{n}{\Rightarrow} w_1w_2 \ldots w_k = w$, kde $X_j \overset{n_j}{\Rightarrow} w_j$ pre $j = 1, 2, \ldots, k$, $0 \leq n_j \leq n$. Podľa indukčného predpokladu ak $X_j \in N$, tak $X_j \in N_{i_j}$ pre nejaké i_j. Ak $X_j \in T$, položíme $i_j = 0$. Z definície množiny N_i teraz vyplýva, že $A \in N_i$, kde $i = 1 + \max(i_1, i_2, \ldots, i_k)$.

Algoritmus 4.1 nám umožňuje zistiť, či gramatika G obsahuje také nadbytočné neterminálne symboly, z ktorých nemožno generovať terminálne reťazce. Sú to prvky množiny $N - N_T$. Množina N_T obsahuje ďalšiu dôležitú informáciu: jazyk generovaný gramatikou G je neprázdny práve vtedy, ak $S \in N_T$.

Príklad 4.7. Majme gramatiku

$$G = (\{A, B, C, D\}, \{a, b, c\}, P, A)$$

kde P obsahuje pravidlá

$A \to aBc \mid CC$
$B \to Bcb \mid DB$
$C \to Cc \mid a$
$D \to DBD \mid e$

Aplikáciou algoritmu 4.1 dostaneme množiny N_i:

$N_0 = \emptyset$
$N_1 = \{C, D\}$
$N_2 = \{A, C, D\}$
$N_3 = N_2 = N_T$

To znamená, že z neterminálneho symbolu B nemožno generovať reťazec terminálnych symbolov. Pretože $A \in N_T$, platí $L(G) \neq \emptyset$.

Prejdeme teraz k problému nájdenia symbolov, ktoré sa nemôžu vyskytnúť v žiadnej vetnej forme gramatiky, a to k problému nájdenia nedostupných symbolov.

Definícia 4.8. Nech $G = (N, T, P, S)$ je gramatika. Označme symbolom V_D množinu dostupných symbolov

$$V_D = \{X/S \overset{*}{\Rightarrow} \alpha X \beta;\ \alpha,\ \beta \in (N \cup T)^*\}$$

Algoritmus 4.2.

Výpočet množiny V_D.

Vstup: gramatika $G = (N, T, P, S)$.

Výstup: množina V_D dostupných symbolov.

Metóda: budeme tvoriť množinu V_0, V_1, ... rekurentne, podobne ako v algoritme 4.1
1. $V_0 := S$, $i := 1$.
2. $V_i := \{X / A \rightarrow \alpha X \beta$ je v $P \wedge A \in V_{i-1}\} \cup V_{i-1}$.
3. Ak $V_i \neq V_{i-1}$, tak $i := i + 1$ a vráť sa na krok 2; inak $V_D := V_i$.

Dôkaz. Pretože $V_i \subseteq N \cup T$, je počet opakovaní kroku 2 konečný. Správnosť algoritmu možno overiť dôkazom ekvivalencie

$X \in V_D$, práve vtedy, keď $S \overset{*}{\Rightarrow} \alpha X \beta$; $\alpha, \beta \in (N \cup T)^*$

úplne analogicky ako v prípade algoritmu 4.1.

Príklad 4.8. Majme gramatiku

$G = (\{A, B, C, D, E\}, \{a, b, c\}, P, A)$

kde P obsahuje pravidlá

$A \rightarrow aC \mid Eb$
$B \rightarrow DD \mid Bb \mid c$
$C \rightarrow E \mid b$
$D \rightarrow aC \mid Bb$
$E \rightarrow aAb \mid b$

Aplikáciou algoritmu 4.2 dostávame množiny V_i

$V_0 = \{A\}$
$V_1 = \{A, a, C, E, b\}$
$V_2 = V_1 = V_D$

Gramatika G teda obsahuje nedostupné symboly c, B, D.

Teraz sformulujeme algoritmus, ktorý odstraňuje všetky nadbytočné symboly gramatiky.

Algoritmus 4.3.
Odstránenie nadbytočných symbolov.
Vstup: gramatika $G = (N, T, P, S)$.
Výstup: gramatika $G' = (N', T', P', S')$, ak $L(G) \neq \emptyset$.
Metóda:
1. Pre gramatiku G vytvor algoritmom 4.1 množinu N_T.
   Ak $S \in N_T$, polož

   $\bar{G} = (N_T, T, \bar{P}, S)$

kde $\bar{P}$ obsahuje iba tie pravidlá z P, ktoré sú vytvorené zo symbolov z $(N_T \cup T)$.

2. Pre gramatiku G vytvor množinu V_D podľa algoritmu 4.2. Teraz polož $G' = (N_T \cap V_D, T \cap V_D, P', S)$, kde P' obsahuje iba tie pravidlá z P, ktoré sú vytvorené zo symbolov z $(N_T \cup T) \cap V_D$. Gramatika G' je výsledná gramatika bez nadbytočných symbolov.

Ak $S \notin N_T$, tak $L(G) = \emptyset$ a všetky symboly gramatiky G sú nadbytočné.

Dôkaz. V kroku 1 algoritmu sú vylúčené neterminálne symboly, z ktorých nemožno generovať terminálne reťazce. V kroku 2 sú odstránené všetky nedostupné symboly. V oboch prípadoch sú spolu so symbolmi odstraňované aj pravidlá obsahujúce nadbytočné symboly. Poradie, v ktorom sa vykonávajú kroky 1 a 2, je dôležité, ako uvidíme v nasledujúcom príklade.

Príklad 4.9. Odstránime nadbytočné symboly z gramatiky z príkladu 4.7. Po aplikácii kroku 1 algoritmu 4.3 získame gramatiku

$\bar{G} = (\{A, C, D\}, \{a, b, c\}, P, A)$

s pravidlami

$A \rightarrow CC$
$C \rightarrow a$
$D \rightarrow e$

V kroku 2 algoritmu 4.3 vytvoríme množinu $V_D = \{A, C, a\}$ dostupných symbolov. Výsledná gramatika G' má tvar

$G' = (\{A, C\}, \{a\}, \{A \rightarrow CC, C \rightarrow a\}, A)$

Keby sme poradie krokov algoritmu 4.3 zamenili, tak aplikáciou kroku 2 na G získame

$V_D = N \cup T$

a teda G sa nezmení, a aplikáciou kroku 1 dostaneme ako výsledok gramatiku $\bar{G}$. Tá však obsahuje nadbytočný symbol D.

4.2.2 Odstránenie *e*-pravidiel

Všeobecný tvar pravidla $A < \alpha$, $\alpha \in (N \cup T)^*$ bezkontextovej gramatiky pripúšťa tiež pravidlá $A \rightarrow e$, kde e je prázdny reťazec, ktoré nazývame *e-pravidlá*. V dôsledku používania e-pravidiel sa môžu skracovať dĺžky vetných foriem tvoriacich deriváciu vety a v špeciálnom prípade môžeme po určitom počte priamych derivácií odvodiť vetnú formu, ktorá sa už v derivácii vety vyskytuje. Gramatika, ktorá dovoľuje vytvárať derivácie s takýmito vlastnosťami, je zjavne viacznačná, a preto proces syntaktickej analýzy (vytváranie derivácie) vety v tejto gramatike môže obsahovať nekonečný cyklus.

Definícia 4.9. Nech $G = (N, T, P, S)$ je gramatika. Ak pre niektorý neterminálny symbol A existuje v G derivácia $A \overset{+}{\Rightarrow} A$, tak hovoríme, že G obsahuje *cyklus*.

Uveďme príklad gramatiky, ktorá obsahuje cyklus v dôsledku použitia e-pravidiel.

Príklad 4.10. Majme gramatiku s pravidlami

$$A \to CBA \mid a$$
$$B \to b \mid e$$
$$C \to c \mid e$$

V tejto gramatike možno vytvoriť deriváciu

$$A \Rightarrow CBA \Rightarrow BA \Rightarrow A$$

a teda $A \overset{+}{\Rightarrow} A$.

Definícia 4.10. Gramatika $G = (N, T, P, S)$ je *gramatikou bez e-pravidiel* vtedy, ak alebo P neobsahuje žiadne e-pravidlo, alebo v P existuje jediné e-pravidlo $S \to e$ a začiatočný symbol S sa nevyskytuje na pravej strane žiadneho pravidla.

Druhá alternatíva definície 4.10 sa vzťahuje na gramatiku G, pre ktorú platí $e \in L(G)$. Zaručuje súčasne, že v žiadnej derivácii v G nedôjde ku skráteniu dĺžky vetnej formy.

Teraz sa budeme zaoberať transformáciou gramatiky na ekvivalentnú gramatiku bez e-pravidiel. Základná myšlienka tejto transformácie vychádza z nasledujúcej úvahy: ak platí, že $B \overset{+}{\Rightarrow} e$ pre neterminál B, tak pravidlo $A \to \alpha B \beta$, $\alpha, \beta \in (\cup T)^*$ možno nahradiť dvojicou pravidiel

$$A \to \alpha B \beta \text{ a } A \to \alpha\beta$$

a postihnúť tak, pre daný výskyt neterminálu B v pôvodnom pravidle, dôsledok derivácie $B \overset{+}{\Rightarrow} e$. Systematickou aplikáciou tejto náhrady na všetky výskyty neterminálov, z ktorých možno generovvať e, možno úplne eliminovať vplyv e-pravidiel gramatiky.

Dôležitým krokom algoritmu odstraňovania e-pravidiel je vytváranie množiny

$$N_e = \{A / A \overset{+}{\Rightarrow} e\}$$

Pretože vytváranie tejto množiny je analogické ako vytváranie množiny N_T podľa algoritmu 4.1, uvedieme iba príslušný rekurentný vzťah:

$$N_i = \{A / A \to \alpha \text{ je v } P \text{ a } \alpha \in N_{i-1}^*\} \cup N_{i-1}$$

kde $N_0 = \emptyset$ ($\emptyset^* = \{e\}$).

Algoritmus 4.4.
Transformácia na gramatiku bez e-pravidiel.
Vstup: gramatika $G = (N, T, P, S)$.
Výstup: ekvivalentná gramatika $G' = (N', T, P', S')$ bez e-pravidiel.
Metóda:

1. Vytvor množinu $N_e = \{A / A \overset{+}{\Rightarrow} e\}$.
2. Množinu pravidiel P' vytváraj takto:
   a) všetky pravidlá $A \to \alpha$, $\alpha \in ((N - N_e) \cup T)^+$ z P sú v P',
   b) pre každé pravidlo

   $$A \to \alpha_0 B_1 \alpha_1 B_2 \dots B_k \alpha_k$$

   $B_i \in N_e$, $\alpha_j \in ((N - N_e) \cup T)^*$ z P pridaj do P' pravidlá $A \to \alpha_0 X_1 \alpha_1 X_2 \dots$ $\dots X_k \alpha_k$, kde $X_i = B_i$, alebo $X_i = e$ pre $i = 1, 2, \dots, k$. V tomto kroku môže vzniknúť e-pravidlo $A \to e$, ak sa reťazec $\alpha_j = e$ pre $j = 0, \dots, k$, alebo pravidlo $A \to A$, ak sa naviac niektorý z neterminálov B_i rovná A. Obe pravidlá sú redundantné, a do množiny P' ich nepridávame, okrem toho je pravidlo $A \to A$ v gramatike zdrojom cyklu,
   c) ak $S \in N_e$ (t. j. $e \in L(G)$), tak pridaj do P' pravidlá $S' \to e \mid S$, kde S' je nový začiatočný symbol.
3. Ak $S \in N_e$, tak $N' = N \cup \{S'\}$, v opačnom prípade $S' = S$ a $N' = N$.

Dôkaz. Gramatika G' je zrejme gramatikou bez e-pravidiel, pretože P' neobsahuje žiadne e-pravidlá okrem pravidla $S' \to e$, v prípade, že $e \in L(G)$, kde S' z tohto pravidla je jediným výskytom nového začiatočného symbolu. Dôkaz ekvivalencie jazykov $L(G)$ a $L(G')$ možno formálne vykonať indukciou [1] s indukčnou hypotézou:

$$A \overset{n}{\underset{G}{\Rightarrow}} w \text{ práve vtedy, keď } A \overset{m}{\underset{G}{\Rightarrow}} w, \text{ pre nejaké } n \geqq m.$$

Príklad 4.11. Gramatiku $G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$ s pravidlami

$$S \to AaBb \mid aCa \mid AC$$
$$A \to BC \mid a \mid e$$
$$B \to Bb \mid b$$
$$C \to cC \mid S \mid e$$

transformujeme na gramatiku bez e-pravidiel.

Najskôr vytvoríme množinu N_e:

$N_0 = \emptyset$ $\quad$ $N_2 = \{S, A, C\}$
$N_1 = \{A, C\}$ $\quad$ $N_3 = N_2 = N_e = \{S, A, C\}$

Pretože $S \in N_e$, bude mať transformovaná gramatika G' tvar

$$G' = (\{S', S, A, B, C\}, \{a, b, c\}, P', S')$$

kde P' obsahuje pravidlá

$S' \to S \mid e$
$S \to AaBb \mid aBb \mid aCa \mid aa \mid AC \mid A \mid C$
$A \to BC \mid B \mid a$
$B \to Bb \mid b$
$C \to cC \mid c \mid S$

4.2.3 Odstránenie jednoduchých pravidiel

Vráťme sa ešte k príkladu 4.11. Vo vstupnej gramatike G existuje derivácia

$$S \Rightarrow AC \Rightarrow C \Rightarrow S$$

a teda cyklus pre neterminál S. Transformáciou gramatiky G na gramatiku G' sme tento cyklus v skutočnosti neodstránili. V G' však už nedochádza k skráteniu dĺžky vetnej formy. Algoritmus 4.4 v prípade, že pre niektoré pravidlo $A \to \alpha$ platí $\alpha \in N_e^+$, vytvára pravidlá tvaru $A \to B$, $B \in N$, ktoré takisto môžu generovať cyklus. Skutočne, v G' z príkladu 4.11 existuje derivácia $S \Rightarrow C \Rightarrow S$ vzhľadom na pravidlá $S \to C$, $C \to S$.

Pravidlá tvaru $A \to B$, $A, B \in N$ nazývame *jednoduché pravidlá*. V gramatike, ktorá neobsahuje e-pravidlá ani jednoduché pravidlá, nemôže vzniknúť cyklus, pretože pre každú priamu deriváciu $\alpha \Rightarrow \beta$ v G platí $|\alpha| \leqq |\beta|$, pričom $|\alpha| = |\beta|$ práve vtedy, ak v derivácii $\alpha \Rightarrow \beta$ bolo použité pravidlo tvaru $A \to a$, $a \in T$.

Princíp všeobecného algoritmu odstraňovania jednoduchých pravidiel ukážeme najskôr na príklade. V gramatike s pravidlami

$A \to aB \mid B$
$B \to bC \mid C$
$C \to c$

sú dve jednoduché pravidlá $A \to B$ a $B \to C$, ktoré vedú k deriváciám

$A \Rightarrow B \Rightarrow C$
$B \Rightarrow C$

V dôsledku týchto derivácií môže mať ekvivalentná gramatika bez jednoduchých pravidiel tvar

$A \to aB \mid bC \mid c$
$B \to bC \mid c$
$C \to c$

Algoritmus 4.5.
Vstup: gramatika $G = (N, T, P, S)$ bez e-pravidiel.
Výstup: ekvivalentná gramatika $G' = (N, T, P', S)$ bez jednoduchých pravidiel.
Metóda:

1. Pre každý terminál A vytvor množinu $N_A = \{B / A \stackrel{*}{\Rightarrow} B\}$ takto:
   a) $N_0 := \{A\}$, $i := 1$,
   b) $N_i := \{C / B \to C$ je pravidlo z P a $B \in N_{i-1}\} \cup N_{i-1}$,
   c) ak $N_i \neq N_{i-1}$ tak $i := i + 1$ a opakuj krok b); v opačnom prípade $N_e := N_i$.
2. Množinu P' vytvor tak, že pre všetky $B \in N_A$ a pre všetky pravidlá $B \to \alpha$, ktoré nie sú jednoduché, budú v P' pravidlá $A \to \alpha$.

Dôkaz. Množina P' obsahuje všetky pravidlá $A \to \alpha$ z P, ktoré nie sú jednoduché, pretože $A \in N_A$ pre všetky A a zrejme neobsahuje žiadne jednoduché pravidlá. Aby sme ukázali, že $L(G) = L(G')$, dokážeme, že platí $L(G') \subseteq L(G)$ a súčasne $L(G) \subseteq L(G')$.

1. $L(G') \subseteq L(G)$
   Nech $w \in L(G')$ a $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$ je derivácia vety w v G'. Ak v priamej derivácii $\alpha_i \underset{G'}{\Rightarrow} \alpha_{i+1}$ bolo použité pravidlo $A \to \beta$, tak v G existuje derivácia $A \stackrel{*}{\Rightarrow} B$ a $B \Rightarrow \beta$, a teda $\alpha_i \underset{G}{\stackrel{+}{\Rightarrow}} \alpha_{i+1}$. Z toho vyplýva, že platí aj $S \underset{G}{\stackrel{*}{\Rightarrow}} w$, a teda $w \in L(G)$.
2. $L(G) \subseteq L(G')$
   Nech $w \in L(G)$ a $S \Rightarrow \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$ je ľavá derivácia vety w v G. Zoberme teraz rastúcu postupnosť $i_1, i_2, \ldots, i_k$ indexov vetných foriem α_j, pre ktoré v ľavej derivácii $\alpha_{j-1} \Rightarrow \alpha_j$ nebolo použité jednoduché pravidlo. Špeciálne platí $i_k = n$. Z konštrukcie gramatiky G' vyplýva, že v G' existuje derivácia

$$S \Rightarrow \alpha_{i_1} \Rightarrow \alpha_{i_2} \Rightarrow \ldots \Rightarrow \alpha_{i_k} = w, \text{ a teda } w \in L(G')$$

Príklad 4.12. Odstránime jednoduché pravidlá v gramatike z príkladu 4.11. Po odstránení e-pravidiel sme získali gramatiku

$S' \to S \mid e$
$S \to AaBb \mid aBb \mid aCa \mid aa \mid AC \mid A \mid C$
$A \to BC \mid B \mid a$
$B \to Bb \mid b$
$C \to cC \mid c \mid S$

Aplikáciou kroku 1 algoritmu 4.5 dostávame

$N_{S'} = \{S', S, A, C, B\}$
$N_S = \{S, A, C, B\}$
$N_A = \{A, B\}$

$N_B = \{B\}$
$N_C = \{C, S, A, B\}$

Aplikáciou kroku 2 získame gramatiku bez jednoduchých pravidiel:

$S' \to e \mid AaBb \mid aBb \mid aCa \mid aa \mid AC \mid BC \mid a \mid Bb \mid b \mid cC \mid c$
$S \to AaBb \mid aBb \mid aCa \mid aa \mid AC \mid BC \mid a \mid Bb \mid b \mid cC \mid c$
$A \to BC \mid a \mid Bb \mid b$
$B \to Bb \mid b$
$C \to cC \mid c \mid AaBb \mid aBb \mid aCa \mid aa \mid AC \mid BC \mid a \mid Bb \mid b$

Táto gramatika už neobsahuje cyklus. Algoritmus odstránenia jednoduchých pravidiel v tomto prípade vedie k nedostupnému neterminálu S. Aj po jeho odstránení počet pravidiel výslednej gramatiky značne vzrástol.

Definícia 4.11. Gramatika G sa nazýva *vlastná*, ak je gramatikou bez nadbytočných symbolov, bez e-pravidiel a bez cyklov.

Veta 4.2. Každú gramatiku možno transformovať na vlastnú gramatiku.
Dôkaz. Dôkaz vyplýva z algoritmov 4.3, 4.4 a 4.5.
Poznamenávame, že definícia vlastnej gramatiky nevylučuje prítomnosť jednoduchých pravidiel. Jednoduché pravidlá môžu byť dokonca výhodné, pretože znižujú celkový počet pravidiel gramatiky. Napríklad gramatika z príkladu 4.4, opisujúca aritmetický výraz, je vlastnou gramatikou. Iba v prípade, že jednoduché pravidlá sú zdrojom cyklov gramatiky, je potrebné ich odstrániť. Algoritmus, ktorý testuje, či gramatika bez e-pravidiel obsahuje cyklus, možno získať jednoduchou modifikáciou kroku 1 algoritmu 4.5 a ponecháme ho ako cvičenie.

4.3 NORMÁLNE TVARY GRAMATIKY

Definícia bezkontextovej gramatiky, ako bola zavedená v Chomského klasifikácii gramatík a jazykov, zdôrazňuje hľadisko všeobecnosti. Požaduje sa iba, aby všetky pravidlá mali tvar $A \to \alpha$, $A \in N$, $\alpha \in (N \cup T)^*$, pričom ani na dĺžku, ani na štruktúru pravej strany α nie sú kladené žiadne obmedzenia.

Vzniká prirodzená otázka, či neexistujú špeciálne tvary prepisovacích pravidiel, ktorých schopnosť reprezentácie jazyka je rovnaká ako pri bezkontextových gramatikách. V teórii formálnych jazykov a gramatík je známy celý rad obmedzení tvaru pravidiel spolu s postupmi, ako transformovať ľubovoľnú bezkontextovú gramatiku na ekvivalentnú gramatiku spĺňajúcu tieto obmedzenia. K najdôležitejším z nich patria dva tvary gramatík nazývané *Chomského* a *Greibachovej normálny tvar* (*forma*) *gramatiky*.

Prv než sa budeme zaoberať transformáciami na normálne tvary a ich vlastnosťami, opíšeme pomocnú transformáciu, ktorú budeme v ďalšom využívať.

Veta 4.3. Nech $G = (N, T, P, S)$ je gramatika a nech $A \to \alpha B \gamma$, $B \in N$ je pravidlo z P a $B \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ sú všetky pravidlá, ktoré majú na ľavej strane neterminál B.
Gramatika $G' = (N, T, P', S)$, kde

$$P' = (P - \{A \to \alpha B \gamma\}) \cup \{A \to \alpha \beta_1 \gamma, A \to \alpha \beta_2 \gamma, \dots, A \to \alpha \beta_n \gamma\}$$

je potom ekvivalentná s gramatikou G.
Dôkaz. Výsledkom tejto transformácie je odstránenie pravidla $A \to \alpha B \gamma$ z gramatiky G tak, že za neterminál B „dosadíme" všetky jeho pravé strany. Formálne sa ukáže platnosť konjunkcie $L(G) \subseteq L(G') \wedge L(G') \subseteq L(G)$.

4.3.1 Chomského normálny tvar gramatiky

Definícia 4.12. Gramatika $G = (N, T, P, S)$ je v Chomského normálnom tvare (skrátene v CNT), ak každé pravidlo z P má jeden z týchto tvarov:
1. $A \to BC$, $A, B, C \in N$ alebo
2. $A \to a$, $a \in T$ alebo
3. $S \to e$, S sa nevyskytuje na pravej strane žiadneho pravidla.

Algoritmus 4.6.
Úprava gramatiky na Chomského normálny tvar.
Vstup: vlastná gramatika $G = (N, T, P, S)$ bez jednoduchých pravidiel.
Výstup: gramatika $G' = (N', T, P', S\uparrow)$ v CNT taká, že $L(G) = L(G|)$.
Metóda: z gramatiky G získame ekvivalentnú gramatiku G' v CNF takto:
1. Množina pravidiel P' obsahuje všetky pravidlá tvaru $A \to a$ z P.
2. Množina pravidiel P' obsahuje všetky pravidlá tvaru $A \to BC$ z P.
3. Ak pravidlo $S \to e$ je v P, tak $S \to e$ je tiež v P'.
4. Pre každé pravidlo tvaru $A \to X_1 \dots X_k$, kde $k > 2$, z P, pridaj do P' túto množinu pravidiel. Symbolom X_i' označíme neterminál X_i, ak $X_i \in N$, alebo nový neterminál, ak $X_i \in T$:
$A \to X_1' \langle X_2 \dots X_k \rangle$
$\langle X_2 \dots X_k \rangle \to X_2' \langle X_3 \dots X_k \rangle$
$\vdots$
$\langle X_{k-1} X_k \rangle \to X_{k-1}' X_k'$
kde každý symbol $\langle X_i \dots X_k \rangle$ je novým neterminálnym symbolom.
5. Pre každé pravidlo tvaru $A \to X_1 X_2$, kde niektorý zo symbolov X_1 alebo X_2 je z T, pridaj do P' pravidlo $A \to X_1' X_2'$.
6. Pre každý nový neterminálny symbol tvaru a' pridaj do P' pravidlo $a' \to a$.

Výsledná gramatika je $G' = (N', T, P', S)$; množina N' obsahuje všetky neterminály z N a nové neterminály tvaru $\langle X_i \dots X_k \rangle$ a a'.

Dôkaz. Gramatika G' zrejme spĺňa podmienky definície CNT. Ekvivalencia gramatík G a G' vyplýva bezprostredne z vety 4.3. Ak odstránime z množiny P' pravidlá, ktoré majú neterminálne symboly tvaru $\langle X_i \dots X_j \rangle$ alebo a', získame gramatiku G.

Veta 4.4. Nech L je bezkontextový jazyk. Potom existuje bezkontextová gramatika G v CNT taká, že $L(G) = L$.

Dôkaz. Predpokladajme, že $L = L(G)$, kde G je bezkontextová gramatika. Podľa vety 4.2 existuje ku G vlastná gramatika. Gramatiku v CNT získame z danej vlastnej gramatiky (prípadnou) aplikáciou algoritmov 4.5 a 4.6.

Príklad 4.13. Na CNT upravíme gramatiku pre aritmetický výraz

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid a$

Najskôr je potrebné odstrániť jednoduché pravidlá

$E \to E + T \mid T * F \mid (E) \mid a$
$T \to T * F \mid (E) \mid a$
$F \to (E) \mid a$

Po aplikácii algoritmu 4.6 získame ekvivalentnú gramatiku v CNF

$$G = (\{E, T, F, \langle + T \rangle, \langle * F \rangle, \langle E) \rangle, (',)', +', *'\}, \{(,), +, *, a\}, P, E)$$

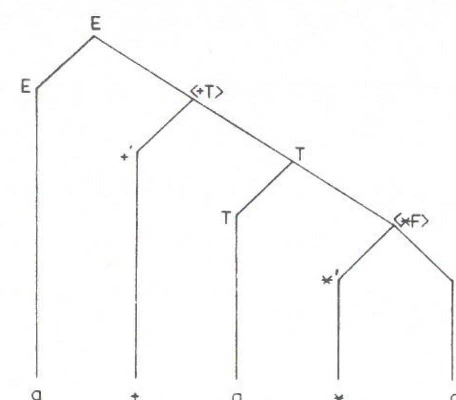
a P obsahuje pravidlá

$E \to E \langle + T \rangle \mid T \langle * F \rangle \mid (' \langle E) \rangle \mid a$
$T \to T \langle * F \rangle \mid (' \langle E) \rangle \mid a$
$F \to (' \langle E) \rangle \mid a$
$\langle + T \rangle \to +' T$
$\langle * F \rangle \to *' F$
$\langle E) \rangle \to E)'$
$(' \to ($
$)' \to)$
$+' \to +$
$*' \to *$

Derivačný strom napr. pre výraz $a + a * a$ je na *obr. 4.7.*

Veta 4.5. Nech G je gramatika v CNT, $w \in L(G)$, $w \neq e$, a nech p je dĺžka derivácie vety w. Potom p je nepárne číslo a platí $|w| = (p + 1)/2$.

Dôkaz. Zoberme podstrom derivačného stromu vety v gramatike v CNT, ktorého koncovými vrcholmi sú bezprostrední predchodcovia koncových



Obr. 4.7. Derivačný strom výrazu $a + a * a$ v gramatike v CNT

vrcholov výsledného derivačného stromu. Tento podstrom je binárnym stromom. Pretože každý binárny strom s n koncovými vrcholmi má $2(n - 1)$ hrán a pretože aplikáciou pravidla tvaru $A \to BC$ vznikajú dve nové hrany, je zrejmé, že stromu, ktorý obsahuje bezprostredných predchodcov koncových vrcholov v derivačnom strome vety w, zodpovedá derivácia dĺžky $|w| - 1$. Celému stromu potom zodpovedá, vzhľadom na aplikáciu iba pravidiel tvaru $A \to a$, derivácia dĺžky

$$p = (|w| - 1) + |w| = 2|w| - 1$$

z čoho vyplýva tvrdenie vety.

4.3.2 Greibachovej normálny tvar gramatiky

Definícia 4.13. Nech $G = (N, T, P, S)$ je gramatika. Neterminálny symbol A z N nazývame rekurzívny ak $A \overset{+}{\Rightarrow} \alpha A \beta$, α, $\beta \in (N \cup T)^*$. Ak $\alpha = e$, tak A je rekurzívny zľava, ak $\beta = e$, A je rekurzívny sprava. Gramatika, ktorá má aspoň jeden rekurzívny neterminálny symbol, sa nazýva *rekurzívna gramatika*, prípadne *gramatika s rekurziou*. Pojmy *gramatika s ľavou a pravou rekurziou* sa definujú analogicky.

Rekurzívnymi gramatikami sa generujú nekonečné jazyky. Napríklad gramatika s pravidlami $S \to Sa \mid a$ generuje jazyk $\{a^i / i > 0\}$. Na druhej strane však gramatiky s ľavou rekurziou, ako uvidíme neskôr, komplikujú vyhľadávanie ľavej derivácie vety. Problém transformácie gramatiky s ľavou rekurziou na ekvivalentnú gramatiku bez ľavej rekurzie je úzko spojený s Greibachovej normálnym tvarom gramatiky, pretože gramatika v tomto tvare je špeciálnym

prípadom gramatiky bez ľavej rekurzie. Preto sa budeme teraz zaoberať problémom odstránenia ľavej rekurzie.

Pre väčšiu názornosť budeme rozlišovať priamu a nepriamu ľavú rekurziu. Ak má gramatika pravidlo tvaru $A \to A\,\alpha$, tak z priamej derivácie $A \Rightarrow A\,\alpha$ vidíme, že je rekurzívna zľava, a v tomto prípade hovoríme o priamej ľavej rekurzii. Zdrojom nepriamej ľavej rekurzie sú napr. pravidlá $A \to B\,\alpha$, $B \to A\,\beta$, pretože vedú k derivácii $A \Rightarrow B\,\alpha \Rightarrow A\,\beta\,\alpha$, a teda $A \overset{+}{\Rightarrow} A\,\beta\,\alpha$.

Definícia 4.14. Nech G je gramatika $G = (N, T, P, S)$. Všetky pravidlá z P, ktoré majú na ľavej strane neterminálny symbol A, nazývame *A-pravidlá* gramatiky G.

Veta 4.6. Nech $G = (N, T, P, S)$ je gramatika a nech

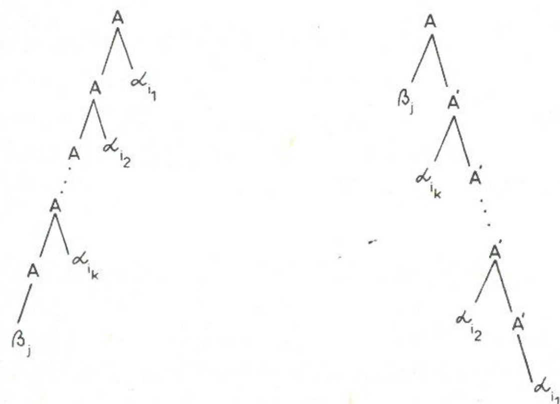
$$A \to A\,\alpha_1 \mid A\,\alpha_2 \mid \dots \mid A\,\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

sú všetky jej A-pravidlá; žiadna pravá strana β_i nezačína symbolom A. Gramatika $G' = (N \cup \{A'\}, T, P', S)$, kde A' je nový neterminálny symbol a P' obsahuje namiesto uvedených A-pravidiel pravidlá

$$A \to \beta_1 \mid \beta_2 \mid \dots \mid B_n \mid \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$
$$A' \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m \mid \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A'$$

je potom ekvivalentná s gramatikou G.

Dôkaz. Opísaná transformácia nahrádza ľavú rekurziu pravou rekurziou. Účinok tejto transformácie je ilustrovaný na odpovedajúcich si derivačných podstromoch v G a G' na *obr. 4.8.* Aby gramatiky G a G' boli ekvivalentné, musia byť množiny reťazcov odvoditeľných z neterminálu A v gramatike G i G' zhodné. Formálny dôkaz zhodnosti týchto množín vykonávať nebudeme.

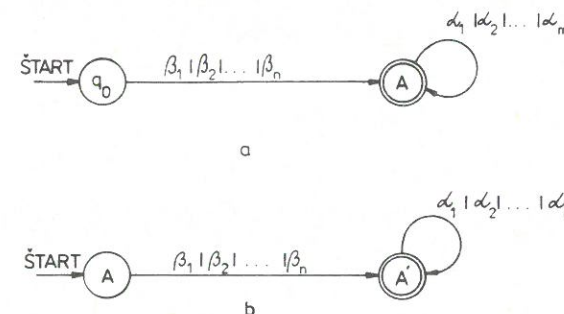


Obr. 4.8. Odstránenie ľavej rekurzie. a) podstrom v G, b) podstrom v G'

Z tvaru pravidiel je však vidieť, že štruktúru reťazcov, odvoditeľných z A v obidvoch gramatických, možno vyjadriť regulárnou množinou

$$\{\beta_1, \beta_2, \dots, \beta_n\} \cdot \{\alpha_1, \alpha_2, \dots, \alpha_m\}^*$$

a opísať konečnými automatmi, ako ukazuje *obr. 4.9.* Automat na *obr. 4.9a* je vytvorený pre A-pravidlá gramatiky G (pre zľava lineárnu regulárnu gramatiku), automat na *obr. 4.9b* zodpovedá pravidlám gramatiky G' (sprava lineárna regulárna gramatika).



Obr. 4.9. a), b) konečné automaty k dôkazu vety 4.6

Príklad 4.14. Aplikujme predchádzajúcu vetu na gramatiku pre aritmetický výraz

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

Pretože táto gramatika obsahuje iba priamu ľavú rekurziu v neterminálach E a T, ako výsledok získame ekvivalentnú gramatiku $G' = (\{E, E', T, T', F\}, \{+, *, (,), a\}, P, E)$ s pravidlami

$E \to T \mid TE'$ $\quad (\alpha_1 = +T, \beta_1 = T)$
$E' \to +T \mid +TE'$
$T \to F \mid FT'$
$T' \to *F \mid *FT'$ $\quad (\alpha_1 = *F, \beta_1 = F)$
$F \to (E) \mid a$

ktorá už nie je zľava rekurzívna.

Všeobecný algoritmus odstránenia ľavej rekurzie spočíva v systematickej substitúcii pravých strán, ktorou upravujeme nepriamu rekurziu na rekurziu priamu, a jej odstraňovaním podľa vety 4.6.

Algoritmus 4.7.

Odstránenie ľavej rekurzie.

Vstup: vlastná gramatika $G = (N, T, P, S)$.

Výstup: gramatika G' bez ľavej rekurzie.

Metóda:

1. Nech $N = \{A_1, A_2, \dots, A_n\}$. Gramatiku budeme transformovať tak, že ak $A_i \to \alpha$ je pravidlo, tak začína alebo terminálom, alebo neterminálom A_j, $j > i$. Za týmto účelom položme $i = 1$.
2. Nech $A_i \to A_i\alpha_1 | \dots | A_i\alpha_m | \beta_1 | \dots | \beta_p$ sú všetky A_i-pravidlá a nech žiadne β_i nezačína neterminálom A_k, ak $k \leqq i$. Teraz nahraď všetky A_i-pravidlá týmito pravidlami:

$$A_i \to \beta_1 | \dots | \beta_p | \beta_1 A_i' | \dots | \beta_p A_i'$$
$$A_i' \to \alpha_1 | \dots | \alpha_m | \alpha_1 A_i' | \dots | \alpha_m A_i'$$

   kde A_i' je nový neterminálny symbol. Takto všetky A_i-pravidlá začínajú alebo terminálom, alebo neterminálom A_k, $k > i$.
3. Ak $i = n$, získali sme výslednú gramatiku G'. V opačnom prípade $i := i + 1$ a $j := 1$.
4. Každé pravidlo tvaru $A_i \to A_j\alpha$ nahraď pravidlami $A_i \to \beta_1\alpha | \dots | \beta_p\alpha$, kde $A_j \to \beta_1 | \dots | \beta_p$ sú všetky A_j-pravidlá. Po tejto transformácii budú všetky A_j-pravidlá začínať alebo terminálom, alebo neterminálom A_k, $k > j$, takže aj všetky A_j-pravidlá budú mať túto vlastnosť.
5. Ak $j = i - 1$, prejdi na krok 2. Inak $j := j + 1$ a prejdi na krok 4.

Dôkaz. Pretože algoritmus aplikuje iba transformácie dané vetami 4.3 a 4.6, platí $L(G') = L(G)$. Formálny dôkaz, že G' je gramatika bez ľavej rekurzie, robiť nebudeme. Treba si však uvedomiť, že krok 2 odstraňuje priamu ľavú rekurziu. Rovnako aj pravidlá, ktoré vzniknú aplikáciou kroku 4 nie sú rekurzívne. Pretože krok 2 sa aplikuje na všetky neterminálne symboly, výsledná gramatika je bez ľavej rekurzie.

Príklad 4.15. Majme gramatiku s pravidlami

$$A \to Ba | c$$
$$B \to CA | bb$$
$$C \to Ab | Bb$$

Napríklad z derivácie

$$A \Rightarrow Ba \Rightarrow CAa \Rightarrow AbAa$$

vidíme, že táto gramatika má všetky neterminálne symboly rekurzívne zľava.

Položme $A = A_1$, $B = A_2$ a $C = A_3$. Jednotlivé kroky algoritmu 4.7 budú mať takýto tvar:

$i = 1$: bez zmeny

$i = 2, j = 1$: bez zmeny

$i = 3, j = 1$, krok 4: pravidlá $C \to Ab | Bb$ nahradíme pravidlami
$C \to Bab | cb | Bb$

$i = 3, j = 2$, krok 4: pravidlá $C \to Bab | cb | Bb$ nahradíme pravidlami
$C \to CAab | bbab | cb | CAb | bbb$

$i = 3, j = 2$, krok 2: odstránime priamu ľavú rekurziu neterminálneho symbolu C a získame výslednú gramatiku bez ľavej rekurzie:

$$A \to Ba | c$$
$$B \to CA | bb$$
$$C \to cb | bbb | bbab | cbC' | bbbC' | bbabC'$$
$$C' \to Aab | Ab | AabC' | AbC'$$

Definícia 4.15. Gramatika $G = (N, T, P, S)$ je v *Greibachovej normálnom tvare* (GNT), ak je bez e-pravidiel a ak každé pravidlo (s prípadnou výnimkou pravidla $S \to e$) má tvar $A \to a\alpha$, kde $a \in T$ a $\alpha \in N^*$.

Lema 4.3. Nech $G = (N, T, P, S)$ je gramatika bez ľavej rekurzie. Potom na množine N existuje také usporiadanie $\prec$, že ak $A \to B\alpha$ je v P, tak $A \prec B$.

Dôkaz. Nech R je taká relácia na N, že $A \, R \, B$ platí práve vtedy, keď $A \overset{*}{\Rightarrow} B\gamma$. Relácia R je zrejme reflexívna a tranzitívna a pretože G je gramatika bez ľavej rekurzie, je relácia R antisymetrická, a preto je aj čiastočným usporiadaním. Každé čiastočné usporiadanie však možno rozšíriť na úplné usporiadanie.

Príklad 4.16. V gramatike bez ľavej rekurzie z príkladu 4.15 má usporiadanie vyhovujúce leme 4.3 tvar

$$C' \prec A \prec B \prec C$$

Ak A je najväčším prvkom v usporiadaní podľa lemy 4.1, tak pravá strana všetkých A-pravidiel začína terminálnym symbolom, čo je zrejmým dôsledkom lemy 4.3.

Algoritmus 4.8.

Transformácia na Greibachovej normálny tvar:

Vstup: vlastná gramatika $G = (N, T, P, S)$ bez ľavej rekurzie.

Výstup: *ekvivalentná gramatika* G′ *v GNT.*

Metóda:

1. Podľa lemy 4.3 vytvor na N také usporiadanie $\prec$, že každé A-pravidlo začína buď terminálom, alebo takým neterminálom B, že $A \prec B$. Nech $N = \{A_1, A_2, \dots, A_n\}$ a $A_1 \prec A_2 \prec \dots \prec A_n$.
2. Polož $i = n - 1$.
3. Ak $i = 0$, prejdi na krok 5, inak nahraď každé pravidlo tvaru $A_i \to A_j\alpha$, kde $j > i$, pravidlami $A_i \to \beta_1\alpha | \dots \dots | \beta_m\alpha$, kde $A_j \to \beta_1 | \dots | \beta_m$ sú A_j-pravidlá. (Každý z reťazcov $\beta_1, \dots, \beta_m$ začína terminálom.)

4. $i := i - 1$ a pokračuj krokom 3.
5. Teraz všetky pravidlá (s výnimkou pravidla $S \to e$) začínajú terminálnym symbolom. V každom pravidle $A \to aX_1 \dots X_k$ nahraď tie symboly X_j, ktoré sú terminálnymi symbolmi, novým neterminálnym symbolom X'_j.
6. Pre všetky X'_j z bodu 5 pridaj do P' pravidlá $X'_j \to X_j$.

Dôkaz. Pretože A_n je v usporiadaní $\prec$ najväčším prvkom, začínajú pravé strany všetkých A_n-pravidiel terminálnymi symbolmi. Krok 3 zaručuje, že túto vlastnosť budú mať aj novozavedené A_i-pravidlá pre $i = 1, 2, \dots, n - 1$. Teraz aplikujeme transformáciu podľa vety 4.3, ktorá zachováva ekvivalenciu gramatík. Kroky 5 a 6 upravujú terminálne symboly pravých strán, ktoré nie sú ich začiatočnými symbolmi, na neterminálne a tiež nemenia gramatikou generovaný jazyk.

Veta 4.7. Nech L je bezkontextový jazyk. Potom existuje gramatika G v GNT taká, že $L = L(G)$.

Dôkaz. Dôkaz vyplýva z lemy 4.3 a z algoritmu 4.8.

Príklad 4.17. Upravme gramatiku aritmetického výrazu na GNT. V príklade 4.14 sme odstránili ľavú rekurziu a získali sme pravidlá

$E \to T \mid TE'$
$E' \to + T \mid + TE'$
$T \to F \mid FT'$
$T' \to * F \mid * FT'$
$F \to (E) \mid a$

Podľa lemy 4.3 zvolíme na množine $\{E, E', T, T', F\}$ usporiadanie, napr. $E \prec E' \prec T \prec T' \prec F$. Po aplikácii kroku 3 získame T-pravidlá $T \to (E) \mid a \mid \mid (E)T' \mid aT'$ a podobne E-pravidlá, ktorých práve strany začínajú terminálnym symbolom. Po zavedení nového neterminálneho symbolu $)'$ budú mať pravidlá výslednej gramatiky v GNT tvar

$E \to (E)' \mid a \mid (E)'T' \mid aT' \mid (E)'E' \mid aE' \mid (E)'T'E' \mid aT'E'$
$E' \to + T \mid + TE'$
$T \to (E)' \mid a \mid (E)'T' \mid aT'$
$T' \to F \mid FT'$
$F \to (E)' \mid a$
$)' \to)$

4.4 SYNTAKTICKÁ ANALÝZA BEZKONTEXTOVÝCH JAZYKOV

Proces vytvárania derivácie či derivačného stromu vety daného jazyka nazývame *rozklad* (rozbor) alebo *syntaktická analýza* vety. Účelom tohto procesu je

nielen zistenie, či reťazec nad danou abecedou jazyka je vetou jazyka, t. j., že neobsahuje syntaktické chyby, ale nemenej dôležité je aj nájdenie syntaktickej štruktúry vety, podľa ktorej možno potom urobiť preklad do iného jazyka. Ako už bolo povedané v úvode, tá časť prekladača, ktorá realizuje syntaktickú analýzu, sa nazýva *syntaktický analyzátor.* Jeden z najväčších praktických prínosov teórie bezkontextových jazykov je práve v oblasti konštrukcie syntaktických analyzátorov, pretože dodáva nielen efektívne algoritmy syntaktickej analýzy, ale umožňuje aj automatizáciu výstavby syntaktických analyzátorov.

Algoritmy syntaktickej analýzy možno rozdeliť podľa spôsobu, ako je vytváraný derivačný strom vety, na dve základné skupiny:
— zhora nadol,
— zdola nahor.

4.4.1 Syntaktická analýza zhora nadol

Pri syntaktickej analýze zhora nadol začíname vytvárať derivačný strom od koreňa — začiatočného symbolu gramatiky a postupnou aplikáciou prepisovacích pravidiel na najľavejší neterminálny symbol získavaných vetných foriem (rezov derivačného stromu) dôjdeme ku koncovým vrcholom derivačného stromu — symbolom analyzovanej vety.

Pri syntaktickej analýze zhora nadol teda vytvárame ľavú deriváciu analyzovanej vety. Výber ľavej derivácie nie je náhodný. Systematická náhrada najľavejšieho neterminálneho symbolu vetnej formy umožňuje v procese syntaktickej analýzy porovnávať generované terminálne symboly vetnej formy so symbolmi analyzovanej vety čo najskôr, ako to vyplýva z vlastností ľavej derivácie.

Príklad 4.18. Majme znovu gramatiku opisujúcu aritmetický výraz. Jej pravidlá pre účely ilustrácie nasledujúcich pojmov očíslujeme takto:

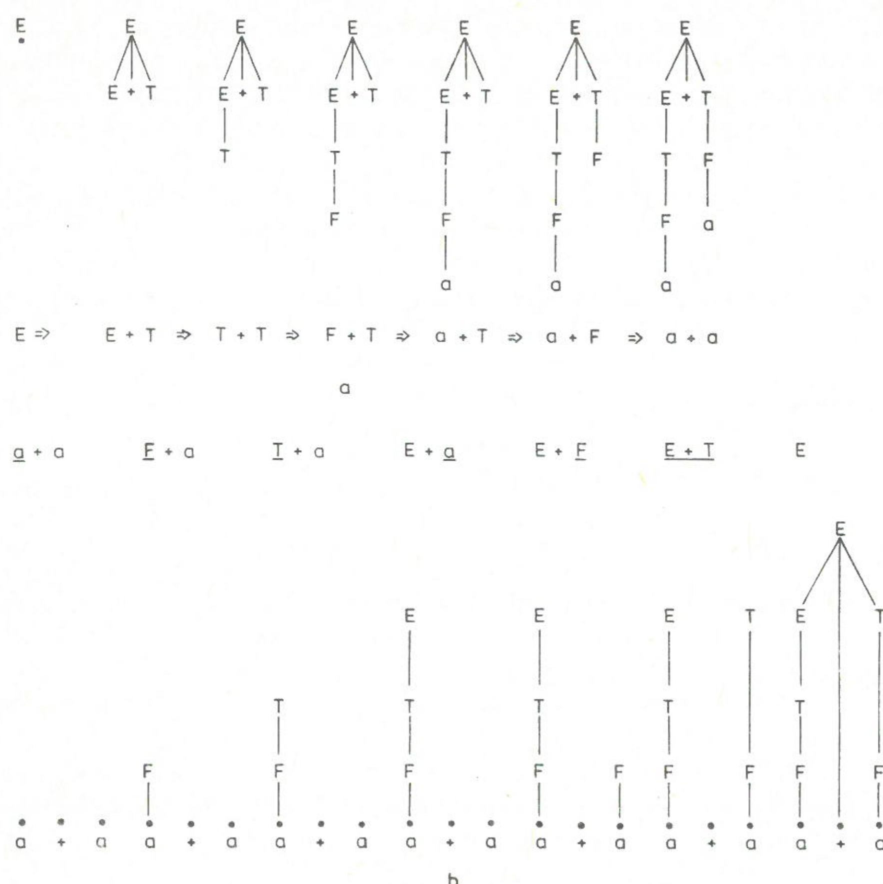
1. $E \to E + T$	4. $T \to F$
2. $E \to T$	5 $T \to (E)$
3. $T \to T * F$	6. $F \to a$

Na *obr. 4.10a* je znázornená syntaktická analýza aritmetického výrazu $a + a$ postupom zhora nadol.

Definícia 4.16. Nech $G = (N, T, P, S)$ je gramatika, ktorej pravidlá sú jednoznačne očíslované prirodzenými číslami $1, 2, \dots, p$, a nech $w \in L(G)$. *Ľavým rozkladom vety w* nazývame postupnosť $p_{i_1} p_{i_2} \dots p_{i_k}$ čísel prepisovacích pravidiel, ktoré boli postupne použité v ľavej derivácii vety w.

Tak napr. v ľavej derivácii

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$$



Obr. 4.10. a) syntaktická analýza zhora nadol, b) syntaktická analýza zdola nahor

v gramatike z príkladu 4.18 boli postupne aplikované prepisovacie pravidlá 1, 2, 4, 6, 4 a 6. Ľavým rozkladom vety $a + a$ je teda postupnosť 124646.

Pri formulácii algoritmu syntaktickej analýzy zhora nadol bude výstupom algoritmu práve ľavý rozklad. Spolu s gramatikou poskytuje rovnakú informáciu ako ľavá derivácia vety, predstavuje však úspornejšiu a praktickejšiu formu reprezentácie výsledku syntaktickej analýzy.

4.4.2 Syntaktická analýza zdola nahor

Pri syntaktickej analýze zdola nahor postupujeme obrátene. Derivačný strom začíname vytvárať od koncových vrcholov — symbolov vety — a postupujeme smerom ku koreňu — začiatočnému symbolu gramatiky. Zatiaľ čo pri analýze

zhora nadol predstavovali jednotlivé kroky priame derivácie, pri analýze zdola nahor postupujeme prostredníctvom priamych redukcií, t. j. náhrady pravej strany pravidla ľavou stranou.

Definícia 4.17. Nech $G = (N, T, P, S)$ je gramatika a nech reťazec $\lambda = \alpha\beta\gamma$, $\alpha, \beta, \gamma \in (N \cup T)^*$, je vetná forma v G. Podreťazec β nazývame *redukčná časť vetnej formy* λ vzhľadom na neterminálny symbol A, ak platí

$$S \overset{*}{\Rightarrow} \alpha A \gamma \Rightarrow \alpha\beta\gamma$$

Najľavejšiu redukčnú časť vetnej formy budeme nazývať *redukčné jadro* alebo jednoducho *jadro* vetnej formy.

Vyhľadávanie redukčných jadier vetnej formy a ich redukcie na zodpovedajúce ľavé strany pravidiel predstavuje základnú akciu syntaktickej analýzy zdola nahor. Na *obr. 4.10b* je tento postup ilustrovaný na príklade analýzy výrazu $a + a$. V získaných vetných formách sú ich redukčné jadrá podčiarknuté.

Pretože od samého začiatku vyberáme k redukcii najľavejšiu redukčnú časť vetnej formy (vety), získavame pri syntaktickej analýze zdola nahor postupnosť vetných foriem, ktorá tvorí v obrátenom poradí pravú deriváciu analyzovanej vety. Skutočne, ak λ_i, λ_{i+1} sú dve susedné vetné formy tejto postupnosti také, že

$$\lambda_i = \alpha\beta\gamma \Leftarrow \alpha A \gamma = \lambda_{i+1}$$

tak $\gamma \in T^*$, β je jadro a A je najpravejší neterminálny symbol vetnej formy λ_{i+1}.

Analogicky ako pojem ľavý rozklad zavedieme ako výstup algoritmu syntaktickej analýzy zdola nahor pojem pravý rozklad.

Definícia 4.18. Nech $G = (N, T, P, S)$ je gramatika, ktorej pravidlá sú očíslované prirodzenými číslami 1, 2, ..., p a nech $w \in L(G)$. *Pravým rozkladom vety* w nazývame obrátenú postupnosť čísel pravidiel, ktorých postupnou aplikáciou získame pravú deriváciu vety w.

Pravý rozklad teda zodpovedá poradiu priamych redukcií pri analýze zdola nahor. Z *obr. 4.10b* a z gramatiky v príklade 4.18 vidíme, že pravým rozkladom vety $a + a$ je postupnosť 642641.

4.4.3 Nedeterministická a deterministická syntaktická analýza

Predpokladajme, že vykonávame syntaktickú analýzu vety $w \in L(G)$ s cieľom nájsť jej ľavý alebo pravý rozklad. Základným problémom algoritmu syntaktickej analýzy je v každom jej kroku rozhodnutie, ktoré pravidlo gramatiky sa má použiť na priamu deriváciu alebo redukciu. Konkrétnejšie pri syntaktickej analýze zhora nadol je potrebné v každej získanej vetnej forme $xA\beta$, $x \in T^*$,

$\beta \in (N \cup T)^*$ určiť, ktoré z A-pravidiel $A \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$ treba aplikovať na najľavejší neterminálny symbol A. Pri syntaktickej analýze zdola nahor je zase potrebné v každej pravej vetnej forme určiť jadro a pravidlo, podľa ktorého sa vykoná redukcia, pričom ani jadro, ani pravidlo na redukciu nemusí byť dané jednoznačne.

Vzniká teda otázka, či existuje algoritmus, ktorý pre každý bezkontextový jazyk L vykoná syntaktickú analýzu reťazca w, určí či $w \in L$, a vytvorí jeho ľavý alebo pravý rozklad, alebo rozhodne, že $w \notin L$.

Takýto všeobecný algoritmus skutočne existuje, má však povahu kombinatorického algoritmu s veľkou výpočtovou zložitosťou. Vychádza zo skutočnosti, že v každom kroku je konečný počet možností, ako pokračovať ďalej. Pri analýze zhora nadol je konečný počet A-pravidiel pre každé $A \in N$, pri analýze zdola nahor je konečný počet potenciálnych jadier a pravidiel na ich redukciu. Algoritmus teda systematicky volí možné alternatívy ďalšieho postupu. Ak sa neskôr ukáže, že výber niektorej alternatívy nevedie k cieľu, je potrebné sa v procese analýzy „vrátiť" a vybrať inú alternatívu. Preto sa takáto analýza nazýva *syntaktická analýza s návratmi*. V najhoršom prípade, po vyčerpaní všetkých možností, syntaktický analyzátor dospeje alebo k hľadanému rozkladu, alebo k poznatku, že analyzovaný reťazec nie je vetou jazyka. Presnú formuláciu algoritmu syntaktickej analýzy s návratmi pre obe triedy analyzátorov možno nájsť napr. v [1]. Tieto analyzátory však majú pre prax malý význam, najmä kvôli časovej a pamäťovej náročnosti. Návraty tiež komplikujú výstavbu prekladačov a problém lokalizácie syntaktických chýb.

Praktickým riešením problému syntaktickej analýzy programovacích jazykov sú *deterministické bezkontextové jazyky*, ktorými sa budeme zaoberať v kapitole 6. Pre tieto jazyky možno vytvoriť syntaktický analyzátor, ktorý dokáže v každom kroku jednoznačne určiť alebo pravidlo na priamu deriváciu, alebo jadro na redukciu, a to podľa nasledujúcich symbolov analyzovanej vety (kontextu podreťazca spracúvanej vetnej formy). Takýto typ syntaktickej analýzy nazývame *deterministická syntaktická analýzy* alebo *syntaktická analýza bez návratov*. Napovieme však, že nie všetky bezkontextové jazyky možno analyzovať deterministickou syntaktickou analýzou.

4.5 VLASTNOSTI BEZKONTEXTOVÝCH JAZYKOV

Už v predchádzajúcich odstavcoch sme sa oboznámili s niektorými dôležitými a charakteristickými vlastnosťami bezkontextových jazykov. Vieme napr., že každý bezkontextový jazyk možno opísať vlastnou bezkontextovou gramatikou alebo gramatikou v Chomského a Greibachovej normálnom tvare, ďalej vieme, že je algoritmicky rozhodnuteľný problém, či reťazec w je alebo nie je prvkom

jazyka $L(G)$. Na druhej strane sme sa zmienili o tom, že problém ekvivalencie dvoch gramatík rovnako ako problém určenia, či je gramatika jednoznačná, sú pre triedu bezkontextových jazykov problémy nerozhodnuteľné.

Teraz sa budeme zaoberať niektorými ďalšími vlastnosťami, ktoré vedú k hlbšiemu chápaniu štruktúry tejto triedy jazykov a v mnohých prípadoch ich možno aplikovať i v oblasti návrhu programovacích jazykov.

Veta 4.8. Problém, či je jazyk generovaný gramatikou neprázdny, je algoritmicky rozhodnuteľný.

Dôkaz. Nech je daná gramatika $G = (N, T, P, S)$. Podľa algoritmu 4.1 vytvoríme množinu N_T. Ak $S \in N_T$, tak $L(G) \neq \emptyset$, inak $L(G) = \emptyset$.

Veta 4.9. Pre každý bezkontextový jazyk L existujú také prirodzené čísla p a q, že ľubovoľnú vetu $w \in L$, pre ktorú platí $|w| > p$, možno vyjadriť v tvare $w = x_1y_1vy_2x_2$, kde

1. $|y_1vy_2| \leqq q$.
2. $y_1y_2 \neq e$.
3. $x_1y_1^ivy_2^ix_2 \in L$ pre všetky $i \geqq 0$.

Dôkaz. Ak $e \in L$, určíme konštanty p a q pre jazyk $L - \{e\}$, ktoré budú zrejme spĺňať podmienku aj pre jazyk L.

Podľa vety 4.4 možno jazyk L generovať gramatikou $G = (N, T, P, S)$ v Chomského normálnom tvare. Najskôr dokážeme že pre deriváciu $A \stackrel{+}{\Rightarrow} \alpha$, $\alpha \in (N \cup T)^+$ v gramatike G platí

$$|\alpha| \leqq 2^{k-1} \tag{4.1}$$

kde k je počet vrcholov najdlhšej cesty v zodpovedajúcom derivačnom strome. Dôkaz urobíme indukciou. Prípad $k = 2$ zodpovedá priamej derivácii $A \Rightarrow \alpha$ a vzhľadom na tvar pravidiel pozostáva reťazec α alebo z jedného (terminálneho) symbolu, alebo z dvoch (neterminálnych) symbolov. Teraz predpokladajme, že dokazovaná nerovnosť platí pre nejaké pevné k a zoberme prípad, keď najdlhšia cesta v derivačnom strome obsahuje $k + 1$ vrcholov. Prvý krok derivácie $A \stackrel{+}{\Rightarrow} \alpha$ potom aplikoval pravidlo tvaru $A \to BC$, B, $C \in N$. Induktívnu hypotézu môžeme teda aplikovať na podstromy s koreňmi B a C, a dostávame tak

$$|\alpha| \leqq 2^{k-1} + 2^{k-1} = 2^{(k+1)-1}$$

Tým sme teda dokázali nerovnosť (4.1).

Nech $|N| = n$. Položme $p = 2^n$ a $q = 2^{n+1}$ a zoberme ľubovoľnú takú vetu $w \in L(G)$, že $|w| > p$. Potom podľa (4.1) najdlhšia cesta C v zodpovedajúcom strome T obsahuje aspoň $n + 2$ vrcholov a aspoň $n + 1$ neterminálnych symbolov. To však znamená, že cesta C obsahuje dva vrcholy r_1, r_2 ohodnotené rovnakým neterminálnym symbolom, povedzme A (obr. 4.11). Vrcholy r_1 a r_2 možno vybrať tak, že podcesta cesty C, začínajúca v r_1, obsahuje najviac $n + 2$

vrcholov. Pretože C je najdlhšia cesta v T, žiadna cesta v T_1, neobsahuje viac ako $n + 2$ vrcholov. Ak y je čelo stromu T_1, tak podľa (4.1) platí

$$|y| \leqq 2^{(n+2)-1} = q$$

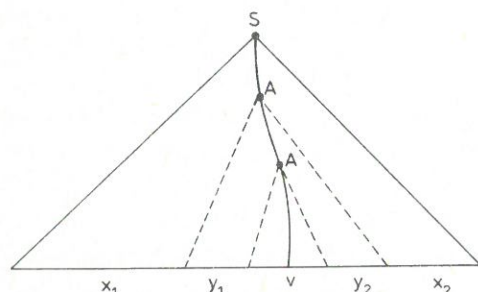
Zoberme ďalej podstrom T_2 stromu T s koreňom r_2. Ak označíme v čelo podstromu T_2, tak $y = y_1 v y_2$ pre nejaké $y_1, y_2 \in T^*$. Naviac reťazce y_1, y_2 nemôžu byť súčasne prázdne, pretože prvé pravidlo v derivácii y muselo mať tvar $A \to BC$, kde B, C sú neterminálne symboly. V gramatike G teda existuje derivácia

$$S \stackrel{*}{\Rightarrow} x_1 A x_2 \stackrel{+}{\Rightarrow} x_1 y_1 A y_2 x_2 \stackrel{+}{\Rightarrow} x_1 y_1 v y_2 x_2 = w$$

To však znamená, že v G existuje aj derivácia

$$S \stackrel{*}{\Rightarrow} x_1 y_1^i v y_2^i x_2 \text{ pre } i \geqq 0$$

Uvedená veta sa nazýva Barova—Hillelova alebo Ogdenova lema



Obr. 4.11. K dôkazu vety 4.9

Pre teóriu bezkontextových jazykov má táto lema veľký význam. Je analógiou vety 3.6 pre regulárne jazyky, pretože opisuje štruktúru nekonečných bezkontextových jazykov. V programovacích jazykoch sú jej odrazom rozličné zátvorkové štruktúry (aritmetických zátvoriek, príkazových zátvoriek, dvojíc **then** — **else** a pod.). Často sa tiež aplikuje pri dôkazoch, že daný jazyk nie je bezkontextový.

Príklad 4.19. Dokážeme, že jazyk $L = \{a^n b^n c^n / n \geqq 1\}$ nie je bezkontextový. Budeme predpokladať opak, a teda existenciu prirodzených čísel p a q, spĺňajúcich podmienku vety 4.9. Zvoľme $k > p/3$ a zoberme vetu $w_1 = a^k b^k c^k$. Vetu w_1 možno potom písať v tvare

$$w_1 = x_1 y_1 v y_2 x_2$$

kde $y_1 y_2 \neq e$ a každá veta $w_i = x_1 y_1^i v y_2^i x_2$ pre $i = 0, 1, 2, \ldots$ patrí do L. Z definície L vidíme, že všetky výskyty písmen a, b, c sú v abecednom poradí, t. j. výskyt b

nepredchádza výskyt a, výskyt c nepredchádza výskyty b, a. Z toho vyplýva, že reťazce y_1 a y_2 obsahujú najviac jeden zo symbolov a, b, c; inak by vo w_i nebola splnená diskutovaná vlastnosť jazyka L. Pretože $y_1 y_2 \neq e$, w_i pre zväčšujúce sa i rastie do nekonečna, ale počet výskytov niektorého zo symbolov a, b alebo c zostáva rovnaký. To teda znamená, že $w_i \in L$ neplatí pre $i \geqq 0$ a zo vzniknutého sporu možno vyvodiť, že L nie je bezkontextový jazyk.

Veta 4.10. Nech L je bezkontextový jazyk a p, q konštanty spĺňajúce podmienky vety 4.9. L je potom nekonečný jazyk práve vtedy, keď obsahuje vetu dĺžky d, $p < d \leqq p + q$.

Dôkaz. Ak jazyk L obsahuje vetu dĺžky $d > p$, tak podľa vety 4.9 je nekonečný. Obrátene predpokladajme, že L je nekonečný. L potom obsahuje vetu w, pre ktorú platí $|w| \geqq p + q$, a platí:

$$w = x_1 y_1 v y_2 x_2,\ y_1 y_2 \neq e,\ |y_1 v y_2| \leqq q,\ x_1 v x_2 \in L$$

V dôsledku toho $p < |x_1 v x_2| < |w|$. Ak $|x_1 v x_2| \leqq p + q$, našli sme vetu požadovanej dĺžky. Ak neplatí $|x_1 v x_2| \leqq p + q$, tak celý postup opakujeme s vetou $x_1 v x_2$. Po konečnom počte krokov dôjdeme k vete dĺžky d, $p < d \leqq p + q$.

Veta 4.10 nám dáva algoritmus, ako testovať, či daný bezkontextový jazyk je nekonečný. Gramatiku G upravíme najskôr na CNT a podľa počtu neterminálnych symbolov určíme konštanty p, q. Na zistenie, či L neobsahuje vetu dĺžky d, kde $p < d \leqq p + q$, stačí preveriť konečný počet derivácií.

Teraz sa zaoberajme vlastnosťami bezkontextových jazykov s ohľadom na uzavrenosť pre niektoré operácie.

Definícia 4.19. Nech $\mathscr{L}$ je trieda jazykov a nech $L \subseteq T^*$ je v triede $\mathscr{L}$. Predpokladajme, že $T = \{a_1, a_2, \ldots, a_n\}$ a že jazyky označené $L_{a_1}, L_{a_2}, \ldots, L_{a_n}$ sú tiež jazyky z $\mathscr{L}$. Trieda $\mathscr{L}$ je uzavretá vzhľadom na substitúciu, ak pre každý výber jazykov $L, L_{a_1}, L_{a_2}, \ldots, L_{a_n}$ je jazyk

$$L' = \{x_1 x_2 \ldots x_n / a_1 a_2 \ldots a_n \in L,\ x_1 \in L_{a_1},\ x_2 \in L_{a_2},\ \ldots,\ x_n \in L_{a_n}\}$$

tiež v triede $\mathscr{L}$.

Príklad 4.20. Nech $L = \{0^n 1^n / n \geqq 1\}$, $L_0 = \{a\}$, $L_1 = \{b^m c^m / m \geqq 1\}$. Substitúciou jazykov L_0 a L_1 za symboly jazyka L získame jazyk

$$L' = \{ab^{m_1}c^{m_1}, aab^{m_2}c^{m_2}b^{m_3}c^{m_3}, \ldots\} =$$
$$= \{a^n b^{m_1} c^{m_1} b^{m_2} c^{m_2} \ldots b^{m_n} c^{m_n} / n \geqq 1,\ m_i \geqq 1,\ 1 \leqq i \leqq n\}$$

Veta 4.11. Trieda bezkontextových jazykov je vzhľadom na substitúciu uzavretá.

Dôkaz. Nech $L \subseteq T^*$ je bezkontextový jazyk, $T = \{a_1, a_2, \ldots, a_n\}$. Nech $L_a \subseteq T_a^*$ je bezkontextový jazyk pre každé $a \in T$. Jazyk získaný substitúciou jazykov L_a za symboly v reťazcoch jazyka L označme L'. Nech $G = (N, T, P, S)$ je gramatika generujúca jazyk L a $G_a = (N_a, T_a, P_a, a')$ sú gramatiky generujúce jazyky L_a. Predpokladajme, že N a N_a sú vzájomne disjuktné. Vytvorme gramatiku $G' = (N', T', P', S)$ takto:

1. $N' = \bigcup_{a \in T} N_a \cup N$.
2. $T' = \bigcup_{a \in T} T_a$; predpokladáme tiež $N' \cap T' = \emptyset$.
3. Nech h je morfizmus na $N \cup T$taký, že $h(A) = A$ pre $A \in N$ a $h(a) = a'$ pre $a \in T$. Nech

$$P' = \{A \to h(\alpha) / A \to \alpha \text{ je v } P\} \cup \bigcup_{a \in T} P_a$$

(P' teda obsahuje pravidlá gramatík G_a a pravidlá gramatiky G, v ktorých sa terminály a zmenia na neterminálne symboly a'.)

Zoberme ľubovoľnú vetu $a_1a_2 \ldots a_n$ z L a vety x_i z L_a pre $1 \leqq i \leqq n$. Potom v G' existuje derivácia

$$S \overset{*}{\Rightarrow} a_1'a_2' \ldots a_n' \overset{*}{\Rightarrow} x_1a_2' \ldots a_n' \overset{*}{\Rightarrow} \ldots \overset{*}{\Rightarrow} x_1 \ldots x_n$$

Teda $L' \subseteq L(G')$. Dôkaz inklúzie $L(G') \subseteq L'$ by si vyžadoval podrobnejšiu analýzu derivačných stromov v G', pre stručnosť výkladu ho nebudeme robiť. Pretože $L' = L(G)$, je L' bezkontextovým jazykom.

Príklad 4.21. V príklade 4.20

$$G = (\{S\}, \{0, 1\}, \{S \to 01, S \to 0S1\}, S)$$
$$G_0 = (\{0'\}, \{a\}, \{0' \to a\}, 0')$$
$$G_1 = (\{1'\}, \{b, c\}, \{1' \to bc, 1' \to b1'c\}, 1')$$

Gramatika G' generujúca jazyk, ktorý vznikol substitúciou jazyka $L(G_0)$ a $L(G_1)$ na $L(G)$, má tvar $G' = (\{S, 0', 1'\}, \{a, b, c\}, P', S)$, kde P' obsahuje pravidlá

$$S \to 0'1' \mid 0'S1'$$
$$0' \to a$$
$$1' \to bc \mid b1'c$$

Veta 4.12. Bezkontextové jazyky sú uzavreté vzhľadom na

1. zjednotenie,
2. zreťazenie,
3. iteráciu $*$,
4. pozitívnu iteráciu $+$,
5. morfizmus.

Dôkaz. Nech L_a, L_b sú jazyky. Substitúciou týchto jazykov:

1. za symboly bezkontextového jazyka $\{a, b\}$ získame jazyk $L_a \cup L_b$,
2. za symboly bezkontextového jazyka $\{ab\}$ získame jazyk L_aL_b,
3. za symboly bezkontextového jazyka a^* získame jazyk L_a^*,
4. za symboly bezkontextového jazyka a^+ získame jazyk L_a^+.
5. Nech $L_a = \{h(a)\}$ pre morfizmus h a každé a z abecedy jazyka L. Substitúciou L_a za symboly bezkontextového jazyka L získame jazyk $h(L)$.

Ak sú L_a a L_b bezkontextové jazyky, podľa vety 4.11 sú aj $L_a \cup L_b$, L_aL_b, L_a^*, L_a^+ a $h(L)$ bezkontextové jazyky.

Veta 4.13. Trieda bezkontextových jazykov nie je uzavretá vzhľadom na prienik, ani vzhľadom na doplnok.

Dôkaz. Majme jazyky $L_1 = \{a^nb^nc^i / n \geqq 1, i \geqq 1\}$ a $L_2 = \{a^ib^nc^n / n \geqq 1, i \geqq 1\}$, ktoré sú oba bezkontextové. Ich prienik $L = L_1 \cap L_2 = \{a^nb^nc^n / n \geqq 1\}$, ako vieme z príkladu 4.19, nie je bezkontextový jazyk.

Pretože každá trieda jazykov, ktorá je uzavretá vzhľadom na zjednotenie a doplnok, musí byť v dôsledku de Morganových zákonov uzavretá vzhľadom na prienik, nemôžu byť bezkontextové jazyky uzavreté vzhľadom na doplnok.

Cvičenia

1. Ukážte, že gramatika s prepisovacím pravidlom $A \to AA$ je viacznačná. Nájdite ekvivalentné pravidlá, ktoré túto vlastnosť odstránia.

2. Daná je gramatika $G = (\{A, B, C, D, E\}, \{+, *, (,), a\}, P, A)$

$$P\colon A \to CB$$
$$B \to +CB \mid e$$
$$C \to ED$$
$$D \to *ED \mid e$$
$$E \to (A) \mid a$$

a) Zistite, či jazyk $L(G)$ je prázdny.
b) Zistite, či G obsahuje nedostupné alebo nadbytočné symboly.
c) Transformujte G na ekvivalentnú gramatiku bez e-pravidiel.
d) Transformujte G na Chomského normálny tvar.
e) Vytvorte derivačný strom, ľavé a pravé odvodenie pre reťazec $(a + a) * a$.

3. Nech $G = (N, T, P, S)$ je bezkontextová gramatika a α_1, α_2 reťazce, pričom $\alpha_1, \alpha_2 \in (N \cup T)^*$. Navrhnite algoritmus, ktorým možno zistiť, či platí $\alpha_1 \underset{G}{\overset{*}{\Rightarrow}} \alpha_2$.

4. Daná je gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, ktorá má pravidlá

$$S \to ba \mid aB$$
$$A \to a \mid aS \mid bAA$$
$$B \to b \mid bS \mid aBB$$

Ukážte, že táto gramatika je viacznačná (zoberte napr. vetu *aabbab*). Pokúste sa nájsť ekvivalentnú jednoznačnú gramatiku.

5. Gramatiku s pravidlami

$S \to A \mid B$
$A \to aB \mid bS \mid b$
$B \to AB \mid bA$
$C \to AS \mid b$

transformujte na ekvivalentnú gramatiku bez nadbytočných symbolov.

6. Nájdite gramatiku bez *e*-pravidiel, ktorá je ekvivalentná s gramatikou

$S \to ABC$
$A \to BB \mid e$
$B \to CC \mid a$
$C \to AA \mid b$

7. Ku gramatike

$S \to A \mid B$
$A \to C \mid D$
$B \to D \mid E$
$C \to S \mid a \mid e$
$D \to S \mid b$
$E \to S \mid c \mid e$

nájdite ekvivalentnú vlastnú gramatiku.

8. Vytvorte algoritmus, ktorým možno zistiť, či gramatika G je alebo nie je gramatikou bez cyklov.

9. V gramatike

$S \to AB$
$A \to BS \mid b$
$B \to SA \mid a$

odstráňte ľavú rekurziu.

10. Na Chomského normálny tvar upravte gramatiku

$G = (\{S, T, L\}, \{a, b, +, -, *, /, [,]\}, P, S)$

kde P obsahuje pravidlá

$S \to T + S \mid T - S \mid T$
$T \to L * T \mid L/T \mid L$
$L \to [S] \mid a \mid b$

11. Gramatiku

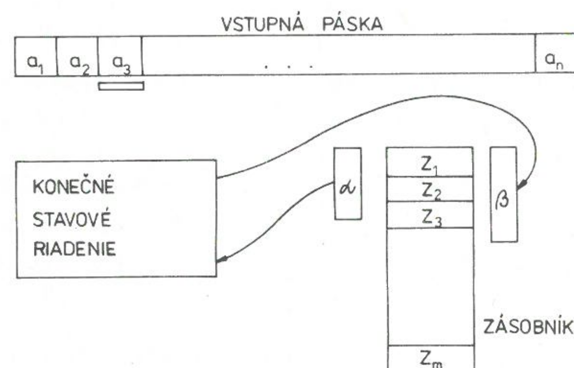
$G = (\{S, A, B\}, \{a, b\}, P, S)$ s pravidlami
$S \to Ba \mid Ab$
$A \to Sa \mid AAb \mid a$
$B \to Sb \mid BBa \mid b$

upravte na Greibachovej normálny tvar.

5 ZÁSOBNÍKOVÉ AUTOMATY

Zásobníkové automaty predstavujú triedu abstraktných matematických strojov, ktoré sú z hľadiska reprezentácie jazykov rovnocenné bezkontextovým gramatikám. Podobne ako konečné automaty v triede regulárnych jazykov zásobníkové automaty umožňujú modelovať syntaktickú analýzu bezkontextových jazykov.

Zásobníkový automat je jednosmerný nedeterministický automat, ktorý má pomocnú, potenciálne nekonečnú pamäť organizovanú ako zásobník. Schéma tohto automatu je uvedená na *obr. 5.1.*



Obr. 5.1. Schéma zásobníkového automatu

Vstupná páska je rozdelená na elementárne záznamy, z ktorých každý obsahuje práve jeden vstupný symbol. Obsah každého záznamu sa postupne číta čítacou hlavou, ale nemožno ho zmeniť. V každom kroku, takte, sa čítacia hlava posunie o jeden záznam vpravo, alebo zostáva na rovnakej pozícii (jednosmerný automat). Konečné stavové riadenie, predpísané prechodovou funkciou automatu, realizuje v závislosti od vnútorného stavu čítaného vstupného symbolu

a reťazca na vrchole zásobníka akciu zmeny vnútorného stavu, posuvu čítacej hlavy a zmeny obsahu zásobníka.

Zásobníkový automat je vo všeobecnosti nedeterministický stroj. Prechodová funkcia môže predpisovať niekoľko alternatívnych akcií, a to vzhľadom na nasledujúci stav a obsah zásobníka, ako aj na to, či sa čítacia hlava posunie na ďalší symbol alebo nie.

5.1 ZÁKLADNÉ DEFINÍCIE

Definícia 5.1. *Zásobníkový automat* je sedmica

$P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$

kde Q je konečná množina vnútorných stavov,
T — vstupná abeceda,
Γ — konečná množina zásobníkových symbolov — zásobníková abeceda,
δ — prechodová funkcia daná zobrazením z množiny $Q \times (T \cup \{e\}) \times \Gamma^*$ do množiny podmnožín množiny $Q \times \Gamma^*$ pričom $\delta(q, a, \alpha) \neq \emptyset$ iba pre konečný počet trojíc (q, a, α),
$q_0 \in Q$ — začiatočný stav,
$Z_0 \in \Gamma$ — začiatočný symbol, určujúci začiatočný obsah zásobníka,
$F \subseteq Q$ — množina koncových stavov.

Definícia 5.2. *Konfiguráciou zásobníkového automatu P* nazývame trojicu (q, w, γ) z množiny $Q \times T^* \times \Gamma^*$, v ktorej
q je momentálny stav stavového riadenia,
w — doteraz neprečítaná časť vstupného reťazca zo vstupnej pásky; prvý symbol reťazca w sa nachádza pod čítacou hlavou. Ak $w = e$, tak boli prečítané všetky symboly zo vstupnej pásky,
γ — obsah zásobníka; ak $\gamma = e$, tak je zásobník prázdny. Obsah zásobníka budeme zapisovať ako reťazec, ktorého najľavejší symbol zodpovedá vrchu zásobníka.

Začiatočná konfigurácia zásobníkového automatu má tvar (q_0, w, Z_0), $w \in T^*$, čo znamená, že automat je v začiatočnom stave q_0, čítacia hlava je nastavená na prvý symbol reťazca w a v zásobníku je začiatočný symbol Z_0.

Koncová konfigurácia má tvar (q, e, γ), $q \in F$, $\gamma \in \Gamma^*$, teda obsah celej vstupnej pásky bol prečítaný a automat je v koncovom stave. Obsah zásobníka môže byť ľubovoľný.

Príklad 5.1. Za predpokladu, že stavové riadenie zásobníkového automatu na *obr. 5.1* je práve v stave r, nachádza sa celý automat v konfigurácii

$(r, a_3a_4 \ldots a_n, Z_1Z_2 \ldots Z_m)$

Definícia 5.3. *Prechod zásobníkového automatu P z jednej konfigurácie do druhej budeme reprezentovať binárnou reláciou* $\vdash_P$ (alebo $\vdash$, ak bude zrejmé, že ide o automat P) na množine všetkých konfigurácií automatu takto:

$$(q, aw, \alpha\gamma) \vdash_P (q', w, \beta\gamma) \overset{\text{df}}{\Leftrightarrow} (q', \beta) \in \delta(q, a, \alpha)$$

$$q, q' \in Q, a \in (T \cup \{e\}), w \in T^*, \alpha, \beta, \gamma \in \Gamma^*$$

Symbolmi $\vdash_P^i$, $\vdash_P^+$, $\vdash_P^*$ budeme označovať postupne *i-tú mocninu*, *tranzitívny*, *reflexívny a tranzitívny uzáver relácie* $\vdash_P$.

Definovanú reláciu $\vdash_P$ interpretujeme takto: Ak sa nachádza zásobníkový automat P v stave q, pod čítacou hlavou je symbol a a na vrchu zásobníka je uložený reťazec α, tak po prečítaní vstupného symbolu a, $a \neq e$, môže automat prejsť do stavu q', čítacia hlava sa posunie o jeden symbol vpravo a na vrch zásobníka sa uloží, po odstránení reťazca α, reťazec β. Ak $\alpha = e$, tak sa zo zásobníka neodstraňuje žiadny symbol, ak $\beta = e$, tak sa žiadny symbol do zásobníka nevkladá.

Ak $a = e$, tak prechod do novej konfigurácie sa neurčuje vstupným symbolom, a čítacia hlava sa preto neposúva. Tento typ prechodu budeme nazývať *e-prechod*. e-prechod môže nastať aj vtedy, keď boli prečítané všetky vstupné symboly.

Definovaný zásobníkový automat je nedeterministický stroj. Ak je automat v konfigurácii $(q, aw, \alpha\gamma)$, $a \neq e$, tak podľa definície 5.1 môže platiť $\delta(q, a, \alpha) = \{(q'_1, \beta_1), (q'_2, \beta_2), \ldots, (q'_k, \beta_k)\}$, a automat teda môže prejsť do ľubovoľnej konfigurácie

$$(q'_i, w, \beta_i\gamma),\ i = 1, 2, \ldots, k$$

Skutočnosť, že $|\delta(q, a, \alpha)| > 1$ však nie je jediným zdrojom nedeterminizmu automatu. Ak $|\delta(q, a, \alpha)| = 1$ a súčasne je definovaná prechodová funkcia pre e-prechod $\delta(q, e, \alpha)$, tak znova nie je jednoznačný prechod automatu do nasledujúcej konfigurácie. Rovnako je to aj v prípade, keď je súčasne definovaná prechodová funkcia $\delta(q, a, \alpha)$ a $\delta(q, a, \eta)$ a η je predpona (prefix) reťazca α. Presnú definíciu deterministického zásobníkového automatu uvedieme v článku 5.4.

Definícia 5.4. Ak pre reťazec w platí relácia $(q_0, w, Z_0) \vdash^* (q, e, \gamma)$ pre nejaké $q \in F$ a $\gamma \in \Gamma^*$, tak hovoríme, že *reťazec w je prijatý* (*akceptovaný*) zásobníkovým automatom P. Množinu $L(P)$ všetkých reťazcov prijatých zásobníkovým automatom P nazývame *jazykom, prijímaným* zásobníkovým automatom P:

$$L(P) = \{w / (q_0, w, Z_0) \vdash^* (q, e, \gamma), q \in F\}$$

Poznámka 5.1. V niektorej literatúre je prechodová funkcia δ automatu P definovaná ako zobrazenie

$$\bar{\delta}: Q \times (T \cup \{e\}) \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$$

Na rozdiel od definície 5.1 sa takýto zásobníkový automat rozhoduje o prechode k nasledujúcej konfigurácii na základe jediného zásobníkového symbolu — vrchu zásobníka. Dá sa ukázať, že obe definície prechodovej funkcie sú ekvivalentné v tom zmysle, že každý zásobníkový automat sa dá upraviť na automat s prechodovou funkciou tvaru $\bar{\delta}$, pričom jazyky prijímané týmito automatmi sú rovnaké. Všeobecnejší tvar prechodovej funkcie výhodne využijeme na opis modelov syntaktických analyzátorov bezkontextových jazykov.

Príklad 5.2. Majme jazyk $L = \{0^n 1^n / n \geqq 0\}$. Zásobníkový automat P, ktorý prijíma jazyk L, je

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{Z, 0\}, \delta, q_0, Z, \{q_0\})$$

kde

$$\begin{aligned}
\delta(q_0, 0, Z) &= \{(q_1, 0Z)\}\\
\delta(q_1, 0, 0) &= \{(q_1, 00)\}\\
\delta(q_1, 1, 0) &= \{(q_2, e)\}\\
\delta(q_2, 1, 0) &= \{(q_2, e)\}\\
\delta(q_2, e, Z) &= \{(q_0, e)\}
\end{aligned}$$

a pracuje tak, že kopíruje všetky nuly zo vstupného reťazca do zásobníka. Ak sa na vstupe objaví symbol 1, tak odstráni jednu nulu zo zásobníka. Okrem toho požaduje, aby všetky nuly predchádzali všetky jednotky. Napríklad pre vstupný reťazec 0011 realizuje automat P tieto prechody:

$$\begin{aligned}
(q_0, 0011, Z) &\vdash (q_1, 011, 0Z)\\
&\vdash (q_1, 11, 00Z)\\
&\vdash (q_2, 1, 0Z)\\
&\vdash (q_2, e, Z)\\
&\vdash (q_0, e, e)
\end{aligned}$$

Ukážeme, že skutočne platí $L(P) = L$. Z definície prechodovej funkcie δ a relácie $\vdash$ vyplýva $(q_0, e, Z) \vdash^0 (q_0, e, Z)$ a teda $e \in L(P)$. Ďalej platí

$$\begin{aligned}
(q_0, 0, Z) &\vdash (q_1, e, 0Z)\\
(q_1, 0^i, 0Z) &\vdash^i (q_1, e, 0^{i+1}Z)\\
(q_1, 1, 0^{i+1}) &\vdash (q_2, e, 0^iZ)\\
(q_2, 1^i, 0^iZ) &\vdash^i (q_2, e, Z)\\
(q_2, e, Z) &\vdash (q_0, e, e)
\end{aligned}$$

čo spolu implikuje

$$(q_0, 0^n1^n, Z) \vdash^{2n+1} (q_0, e, e) \text{ pre } n \geqq 1$$

a teda $L \subseteq L(P)$.

Dokázať obrátenú inklúziu $L(P) \subseteq L$, t. j. dokázať, že zásobníkový automat neprijíma iné reťazce než 0^n1^n, $n \geqq 1$, je ťažšie. Ak prijíma zásobníkový automat P vstupný reťazec, tak musí prejsť postupnosťou stavov q_0, q_1, q_2, q_0. Ak platí $(q_0, w, Z) \vdash^i (q_1, e, \gamma)$, tak $w = 0^i$ a $\gamma = 0^iZ$. Podobne ak $(q_2, w, \alpha) \vdash^i (q_0, e, \beta)$, tak $w = 1^i$ a $\alpha = 0^i\beta$. Z definície δ vyplýva, že relácia $(q_1, w, \alpha) \vdash (q_2, e, \beta)$ platí práve vtedy, keď $w = 1$ a $\alpha = 0\beta$, a relácia $(q_2, w, Z) \vdash^* (q_0, e, e)$ platí práve vtedy, keď $w = e$. To znamená, že ak platí $(q_0, w, Z) \vdash^i (q_0, e, \gamma)$ pre nejaké $i > 0$, tak $w = 0^n1^n$, $i = 2n + 1$ a $\gamma = e$, teda $L(P) \subseteq L$.

Príklad 5.3. Majme zásobníkový automat P:

$$P = (\{q, p\}, \{a, b\}, \{a, b, S, Z\}, \delta, q, Z, \{p\})$$

kde zobrazenie δ je definované takto:

$\delta(q, a, e) = \{(q, a)\}$
$\delta(q, b, e) = \{(q, b)\}$
$\delta(q, e, e) = \{(q, S)\}$
$\delta(q, e, aSa) = \{(q, S)\}$
$\delta(q, e, bSb) = \{(q, S)\}$
$\delta(q, e, SZ) = \{(p, e)\}$

Dá sa ukázať, podobne ako v predchádzajúcom príklade, že automat P prijíma jazyk

$$L = \{ww^R / w \in \{a, b\}^*\}$$

Automat P pracuje tak, že najskôr presunie do zásobníka predponu w vstupného reťazca ww^R a potom na vrch zásobníka uloží symbol S označujúci stred. Ďalej po presune každého symbolu reťazca w^R do zásobníka nahrádza vrchný reťazec aSa alebo bSb symbolom S. Vstupný reťazec prijme vtedy, keď po jeho prečítaní zostane v zásobníku začiatočný symbol Z (prechodom do koncového stavu). Tak napr. pre vstupný reťazec $aabbaa$ môže automat P vykonať tieto prechody:

$(q, aabbaa, Z) \vdash (q, abbaa, aZ)$
$\vdash (q, bbaa, aaZ)$
$\vdash (q, baa, baaZ)$
$\vdash (q, baa, SbaaZ)$
$\vdash (q, aa, bSbaaZ)$

$\vdash (q, aa, SaaZ)$
$\vdash (q, a, aSaaZ)$
$\vdash (q, a, SaZ)$
$\vdash (q, e, aSaZ)$
$\vdash (q, e, SZ)$
$\vdash (p, e, e)$

Uvedený automat je nedeterministický. Hodnotami prechodovej funkcie δ sú síce jednoprvkové množiny, ale nie je jednoznačne určené, či sa má ďalší vstupný symbol uložiť do zásobníka, alebo či sa má vykonať niektorý z e-prechodov.

5.2 MODELY SYNTAKTICKÝCH ANALYZÁTOROV BEZKONTEXTOVÝCH JAZYKOV

Modely syntaktického analyzátora bezkontextového jazyka reprezentované zásobníkovým automatom predstavujú dôležitú abstrakciu umožňujúcu pochopiť a riešiť problém konštrukcie riadiacej časti prekladačov programovacích jazykov. Ukážeme, ako možno ku každej bezkontextovej gramatike G vytvoriť zásobníkové automaty R a R', ktoré sú ekvivalentné s gramatikou G v tom zmysle, že definujú rovnaký jazyk ako gramatika G a naviac prijmú vstupný reťazec z $L(G)$ práve vtedy, ak postupnosť prechodov medzi ich konfiguráciami zodpovedá ľavej, resp. pravej derivácii vstupného reťazca v gramatike G. Tieto zásobníkové automaty budeme považovať za prirodzené modely syntaktickej analýzy zhora nadol, resp. zdola nahor.

Veta 5.1. Nech $G = (N, T, P, S)$ je bezkontextová gramatika a nech R je zásobníkový automat

$$R = (\{p, q, r\}, T, N \cup T \cup \{\#\}, \delta, p, \#, \{r\}), \ \# \notin (N \cup T)$$

ktorého prechodová funkcia δ je definovaná týmto predpisom:

1. $\delta(p, e, e) = \{(q, S)\}$.
2. $\delta(q, e, A)$ obsahuje (q, β) práve vtedy, keď je v P pravidlo $A \to \beta$.
3. $\delta(q, a, a) = \{(q, e)\}$ pre všetky $a \in T$.
4. $\delta(q, e, \#) = \{(r, e)\}$.

Potom $L(G) = L(R)$ a zásobníkový automat modeluje syntaktickú analýzu jazyka $L(G)$ metódou zhora nadol.

Dôkaz. Zásobníkový automat pracuje tak, že vytvára deriváciu vstupného reťazca. Po uložení začiatočného symbolu S na vrch zásobníka (podľa 1) vykonáva postupne alebo expanziu neterminálneho symbolu na vrchu zásobníka (podľa 2), alebo porovnanie derivovaného terminálneho symbolu na vrchu zásobníka s čítaným symbolom vstupnej vety a jeho odstránenie zo zásobníka (podľa 3). Po odstránení všetkých symbolov gramatiky zo zásobníka prejde

automat do koncového stavu (podľa 4). V konfigurácii automatu R, keď je na vrchu zásobníka neterminál, je vetná forma hľadanej derivácie daná zreťazením spracovanej časti vstupného reťazca a obsahu zásobníka. Pretože symbol na vrchu zásobníka je jej najľavejším neterminálom, automat R vytvára ľavú deriváciu a modeluje tak syntaktickú analýzu zhora nadol. Dokážme, že automat R prijíma práve jazyk $L(G)$. Najskôr ukážeme, že

$$A \stackrel{+}{\Rightarrow} w \text{ vtedy a len vtedy, ak } (q, w, A) \vdash^{+} (q, e, e)$$

Nutnú podmienku dokážeme indukciou pre m. Predpokladajme, že $A \stackrel{m}{\Rightarrow} w$. Ak $m = 1$ a $w = a_1 \dots a_k$, $k \geqq 0$, tak

$$(q, a_1 \dots a_k, A) \vdash (q, a_1 \dots a_k, a_1 \dots a_k) \vdash^{k} (q, e, e)$$

Predpokladajme teraz, že pre všetky $m \leqq l$ platí tvrdenie, ak $A \stackrel{m}{\Rightarrow} w$, tak $(q, w, A) \vdash^{+} (q, e, e)$. Dokážeme, že toto tvrdenie platí aj pre $m = l + 1$. Nech $A \stackrel{l+1}{\Rightarrow} w$. Prvý krok tejto derivácie má tvar $A \Rightarrow X_1 X_2 \dots X_k$, pričom $X_i \stackrel{m_i}{\Rightarrow} x_i$ pre $m_i \leqq l$, $1 \leqq i \leqq k$ a $x_1 x_2 \dots x_k = w$. Teda

$$(q, w, A) \vdash (q, w, X_1 \dots X_k)$$

Ak $X_i \in N$, tak podľa indukčného predpokladu

$$(q, x_i, X_i) \vdash^{*} (q, e, e)$$

Ak $X_i = x_i$, $x_i \in T$, tak

$$(q, x_i, X_i) \vdash (q, e, e)$$

Teraz už vidíme, že ľavej derivácii

$$A \to X_1 \dots X_k \stackrel{m_1}{\Rightarrow} x_1 X_2 \dots X_k \stackrel{m_2}{\Rightarrow} \dots \stackrel{m_k}{\Rightarrow} x_1 x_2 \dots x_k = w$$

zodpovedá táto postupnosť prechodov:

$$(q, x_1 \dots x_k, A) \vdash (q, x_1 \dots x_k, X_1 \dots X_k) \vdash^{*}$$
$$\vdash^{*} (q, x_2 \dots x_k, X_2 \dots X_k) \vdash^{*} \dots \vdash^{*} (q, e, e)$$

Obrátenú implikáciu, ak $(q, w, A) \vdash^{n} (q, e, e)$, tak $A \stackrel{+}{\Rightarrow} w$, dokážeme indukciou pre n. Pre $n = 1$ sa $w = e$ a $A \to e$ je pravidlo v P. Predpokladajme, že indukčný predpoklad platí pre všetky $n \leqq l$; dokážeme, že platí tiež pre $n = l + 1$.

Nech $(q, w, A) \vdash^{l+1} (q, e, e)$ a $w = x_1 x_2 \dots x_k$. Potom prvý prechod automatu má tvar $(q, w, A) \vdash (q, w, X_1 \dots X_k)$ a platí $(q, x_i, X_i) \vdash^{n_i} (q, e, e)$, $n_i \leqq l$ pre $1 \leqq i \leqq k$. Prvému prechodu zodpovedá pravidlo $A \to X_1 X_2 \dots X_n$ gramatiky G

a platí $X_i \stackrel{+}{\Rightarrow} x_i$ (z indukčného predpokladu) alebo $X_i = x_i$, ak $X_i \in T$. Existuje teda ľavá derivácia

$$A \Rightarrow X_1 \dots X_k \stackrel{*}{\Rightarrow} x_1 X_2 \dots X_k \stackrel{*}{\Rightarrow} \dots \stackrel{*}{\Rightarrow} x_1 x_2 \dots x_{k-1} X_k \stackrel{*}{\Rightarrow}$$
$$\stackrel{*}{\Rightarrow} x_1 x_2 \dots x_k = w \text{ a platí } A \stackrel{*}{\Rightarrow} w$$

Ak položíme $A = S$, tak vzhľadom na dokazované tvrdenie a vzhľadom na jednoznačné prechody 1 a 4 platí

$$S \stackrel{+}{\Rightarrow} w \text{ práve vtedy, ak}$$

$$(p, w, \#) \vdash (q, w, S\#) \vdash^{*} (q, e, \#) \vdash (r, e, e)$$

a teda $L(R) = L(G)$.

Príklad 5.4. Majme bezkontextovú gramatiku

$$G = (\{R, K, I\}, \{a, \varepsilon, +, *, (,)\}, P, R)$$

s pravidlami

$$R \to R + K \mid K$$
$$K \to KI \mid I$$
$$I \to I^* \mid (R) \mid a \mid \varepsilon$$

Táto gramatika opisuje regulárne výrazy, vytvorené zo symbolov a, ε. Symbol $+$ označuje zjednotenie, operátor zreťazenia sa explicitne neuvádza, $*$ označuje interáciu a ε označuje prázdny reťazec (je potrebné použiť iný symbol než e, aby pravidlo $I \to \varepsilon$ nechápalo ako e-pravidlo).

Zásobníkový automat, ktorý podľa vety 5.2 zodpovedá tejto gramatike, má tvar

$$R = (\{p, q, r\}, \{a, \varepsilon, +, *, (,)\}, \{R, K, I, a, \varepsilon, +, *, (,), \#\}, p, \#, \{r\})$$

$$\delta(p, e, e) = \{(q, R)\}$$
$$\delta(q, e, R) = \{(q, R + K), (q, K)\}$$
$$\delta(q, e, K) = \{(q, KI), (q, I)\}$$
$$\delta(q, e, I) = \{(q, I^*), (q, (R)), (q, a), (q, \varepsilon)\}$$
$$\delta(q, b, b) = \{(q, e) \text{ pre } b \in \{a, \varepsilon, +, *, (,)\}$$
$$\delta(q, e, \#) = \{(r, e)\}$$

Napríklad pre vstupný reťazec $\varepsilon + a^*$ môže zásobníkový automat R realizovať túto postupnosť prechodov:

$$(p, \varepsilon + a^*, \#) \vdash (q, \varepsilon + a^*, R\#)$$
$$\vdash (q, \varepsilon + a^*, R + K\#)$$
$$\vdash (q, \varepsilon + a^*, K + K\#)$$
$$\vdash (q, \varepsilon + a^*, I + K\#)$$

$$\begin{aligned}&\vdash (q,\ \varepsilon + a^*,\ \varepsilon + K\ \#)\\&\vdash (q,\ + a^*,\ + K\ \#)\\&\vdash (q,\ a^*,\ K\ \#)\\&\vdash (q,\ a^*,\ I\ \#)\\&\vdash (q,\ a^*,\ I^*\ \#)\\&\vdash (q,\ a^*,\ a^*\ \#)\\&\vdash (q,\ *,\ *\ \#)\\&\vdash (q,\ e,\ \#)\\&\vdash (r,\ e,\ e)\end{aligned}$$

Uvedenej postupnosti prechodov zodpovedá v gramatike G ľavá derivácia reťazca $\varepsilon + a^*$:

$$R \Rightarrow R + K \Rightarrow K + K \Rightarrow I + K \Rightarrow \varepsilon + K \Rightarrow \varepsilon + I \Rightarrow \varepsilon + I^* \Rightarrow \varepsilon + a^*$$

Veta 5.2. Nech $G = (N, T, P, S)$ je bezkontextová gramatika a nech R' je zásobníkový automat

$$R' = (\{q, r\},\ T,\ N \cup T \cup \{\#\},\ \delta,\ q,\ \#,\ \{r\}),\quad \# \notin (N \cup T)$$

ktorého prechodová funkcia δ je definovaná týmto predpisom:

1. $\delta(q, a, e) = \{(q, a)\}$ pre všetky $a \in T$.
2. $\delta(q, e, \alpha^R)$ obsahuje (q, A) práve vtedy, ak v P je pravidlo $A \rightarrow \alpha$.
3. $\delta(q, e, S\ \#) = \{(r, e)\}$.

Potom $L(G) = L(R')$ a zásobníkový automat R' modeluje syntaktickú analýzu jazyka $L(G)$ metódou zdola nahor.

Dôkaz. Ukážeme, že automat R' vytvára pravú deriváciu vstupného reťazca postupnými redukciami jadra pravých vetných foriem. Automat realizuje dva typy akcií; presun terminálnych symbolov do zásobníka (podľa 1) a redukciu jadra (pravej strany pravidla) umiestneného na vrchu zásobníka na príslušný neterminál na ľavej strane pravidla (podľa 2). V okamihu, keď je celý vstupný reťazec zredukovaný na začiatočný symbol, prejde automat R' do koncového stavu. Každá získaná pravá vetná forma γAy, $\gamma \in (N \cup T)^*$, $A \in N$, $y \in T^*$, je v okamihu po vykonaní redukcie uložená v zásobníku (reťazec γA, A je vrch) a na doteraz neprečítanej časti vstupnej pásky (reťazec y).

Ukážeme, že automat R' prijíma práve jazyk $L(G)$. V tomto dôkaze nebudeme explicitne uvádzať, že $\Rightarrow$ označuje pravú deriváciu.

Indukčný predpoklad, ktorý dokážeme indukciou pre n, má tvar

$$S \overset{*}{\Rightarrow} \gamma Ay \overset{n}{\Rightarrow} xy \text{ implikuje } (q,\ xy,\ \#) \overset{*}{\vdash} (q,\ y,\ A\gamma^R\ \#)$$

Pre $n = 0$ táto hypotéza platí, pretože $\gamma A = x$, a teda automat R vykonáva prechody typu 1 z definície funkcie δ, presúvajúce reťazec x do zásobníka.

Teraz predpokladajme, že indukčný predpoklad platí pre všetky pravé derivácie kratšie než n a platí

$$\gamma Ay \Rightarrow \gamma\alpha y \overset{n-1}{\Rightarrow} xy$$

Ak reťazec $\gamma\alpha$ pozostáva iba z terminálnych symbolov, tak $\gamma\alpha = x$ a $(q,\ xy,\ \#) \overset{*}{\vdash} (q,\ y,\ \alpha^R\gamma^R\ \#)$. Ak neplatí $\gamma\alpha \in T^*$, tak $\gamma\alpha = \beta Bz$, $B \in N$, $z \in T^*$, a podľa indukčného predpokladu derivácia

$$S \overset{*}{\Rightarrow} \beta Bzy \overset{n-1}{\Rightarrow} xy$$

implikuje platnosť relácie

$$(q,\ xy,\ \#) \overset{*}{\vdash} (q,\ zy,\ B\beta^R\ \#)$$

Platí teda $(q,\ zy,\ B\beta^R) \overset{*}{\vdash} (q,\ y,\ z^R B\beta^R\ \#) \vdash (q,\ y,\ A\gamma^R\ \#)$. Tým sme dokázali indukčný predpoklad a pretože

$$(q,\ e,\ S\ \#) \vdash (r,\ e,\ e)$$

platí $L(G) \subseteq L(R')$.

Aby sme dokázali, že aj $L(R') \subseteq L(G)$, ukážeme, že platí implikácia:

$$\text{ak } (q,\ xy,\ \#) \overset{n}{\vdash} (q,\ y,\ A\gamma^R\ \#), \text{ tak } \gamma Ay \overset{*}{\Rightarrow} xy$$

Pre $n = 0$ táto implikácia zjavne platí. Predpokladajme, že platí tiež pre všetky postupnosti prechodov, kratšie než n. Pretože na vrchu zásobníka je neterminálny symbol, vykonal sa posledný prechod podľa typu 2 v definícii funkcie prechodu δ, a môžeme teda písať

$$(q,\ xy,\ \#) \overset{n-1}{\vdash} (q,\ y,\ \alpha^R\gamma^R\ \#) \vdash (q,\ y,\ A\gamma^R\ \#)$$

kde $A \rightarrow \alpha$ je pravidlo z P. To však znamená, že v G existuje derivácia $\gamma Ay \Rightarrow \gamma\alpha y \overset{*}{\Rightarrow} xy$.

Ako špeciálny prípad uvažujme implikáciu:

$$\text{ak } (q,\ w,\ \#) \overset{*}{\vdash} (q,\ e,\ S\ \#), \text{ tak } S \overset{*}{\Rightarrow} w$$

Pretože platí $w \in L(R')$ práve vtedy, ak $(q,\ w,\ \#) \overset{*}{\vdash} (q,\ e,\ S\ \#) \vdash (r,\ e,\ e)$, tak $L(R') \subseteq L(G)$ a spolu s predchádzajúcou časťou dôkazu dostávame $L(G) = L(R')$.

Príklad 5.5. Pre gramatiku

$$G = (\{R, K, I\},\ \{a, \varepsilon, +, *, (,)\},\ P,\ R)$$

s pravidlami

$$R \to R + K \mid K$$
$$K \to KI \mid I$$
$$I \to I^* \mid (R) \mid a \mid \varepsilon$$

vytvoríme zásobníkový automat R' modelujúci syntaktickú analýzu zdola nahor:

$$R' = (\{q, r\}, \{a, \varepsilon, +, *, (,)\}, \{R, K, I, a, \varepsilon, +, *, (,), \#\}, \delta, q, \#, \{r\})$$
$$\delta(q, b, e) = \{(q, b)\} \text{ pre všetky } b \in \{a, \varepsilon, +, *, (,)\}$$
$$\delta(q, e, K + R) = \{(q, R)\}$$
$$\delta(q, e, K) = \{(q, R)\}$$
$$\delta(q, e, IK) = \{(q, K)\}$$
$$\delta(q, e, I) = \{(q, K)\}$$
$$\delta(q, e, *I) = \{(q, I)\}$$
$$\delta(q, e,) R () = \{(q, I)\}$$
$$\delta(q, e, a) = \{(q, I)\}$$
$$\delta(q, e, \varepsilon) = \{(q, I)\}$$
$$\delta(q, e, R \#) = \{(r, e)\}$$

Napríklad pre vstupný reťazec $\varepsilon + a^*$ môže zásobníkový automat R' vykonať postupnosť prechodov

$$(q, \varepsilon + a^*, \#) \vdash (q, + a^*, \varepsilon \#)$$
$$\vdash (q, + a^*, I \#)$$
$$\vdash (q, + a^*, K \#)$$
$$\vdash (q, + a^*, R \#)$$
$$\vdash (q, a^*, + R \#)$$
$$\vdash (q, *, a + R \#)$$
$$\vdash (q, *, I + R \#)$$
$$\vdash (q, e, *I + R \#)$$
$$\vdash (q, e, I + R \#)$$
$$\vdash (q, e, K + R \#)$$
$$\vdash (q, e, R \#)$$
$$\vdash (r, e, e)$$

Táto postupnosť zodpovedá pravej derivácii

$$R \Rightarrow R + K \Rightarrow R + I \Rightarrow R + I^* \Rightarrow R + a^* \Rightarrow K + a^* \Rightarrow I + a^* \Rightarrow \varepsilon + a^*$$

5.3 EKVIVALENCIA JAZYKOV PRIJÍMANÝCH ZÁSOBNÍKOVÝMI AUTOMATMI A BEZKONTEXTOVÝCH JAZYKOV

Vety 5.1 a 5.2 z predchádzajúceho článku ukazujú, že medzi zásobníkovými automatmi a bezkontextovými gramatikami existuje veľmi úzky vzťah, ktorý dovoľuje ku každej bezkontextovej gramatike vytvoriť, z hľadiska špecifikácie formálnych jazykov, ekvivalentný zásobníkový automat. Teraz ukážeme, že trieda jazykov prijímaných zásobníkovými automatmi neobsahuje iné jazyky ako bezkontextové, a teda, že jazyky prijímané zásobníkovými automatmi a bezkontextové jazyky sú identické triedy formálnych jazykov.

Aby sme uľahčili konštrukciu ekvivalentnej bezkontextovej gramatiky k ľubovoľnému zásobníkovému automatu, zavedieme najskôr dva varianty zásobníkových automatov, ktoré sa líšia od definície 5.1 v definícii prechodovej funkcie δ a v definícii koncovej konfigurácie. Nebudeme uvádzať úplné dôkazy ekvivalencie týchto zásobníkových automatov, ale sa obmedzíme iba na ich konštrukciu.

Veta 5.3. Nech $P = (Q, R, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Potom existuje zásobníkový automat $P_1 = (Q_1, T, \Gamma_1, \delta_1, q_1, Z_1, F_1)$, kde δ_1 je funkcia tvaru

$$\delta_1 : Q_1 \times (T \cup \{e\}) \times \Gamma_1 \to 2^{(Q_1 \times \Gamma_1^*)}$$

taký že $L(P) = L(P_1)$.

Dôkaz. Automat P_1 sa líši od automatu P tým, že jeho prechod do bezprostredne nasledujúcej konfigurácie je určovaný vrchom zásobníka, a nie reťazcom na vrchu zásobníka. Ak $(q', \beta) \in \delta_1(q, a, Z)$, tak platí relácia

$$(q, aw, Z\gamma) \underset{P_1}{\vdash} (q', w, \beta\gamma),\ q, q' \in Q_1,\ a \in (T \cup \{e\}),\ Z \in \Gamma_1,\ \beta, \gamma \in \Gamma_1^*$$

medzi konfiguráciami automatu P_1. Automat P_1 má teda v porovnaní s automatom P obmedzenejšie možnosti rozhodovania na základe obsahu zásobníka. Ak má mať celkove rovnakú rozhodovaciu silu, tak je potrebné rozšíriť množinu vnútorných stavov automatu P tak, aby niesli aj informáciu o reťazci na vrchu zásobníka automatu P. Položme

$$k = \max \{|\alpha| / \delta(q, a, \alpha) \neq \emptyset,\ q \in Q,\ a \in (T \cup \{e\}),\ \alpha \in \Gamma^*\}$$

Stavy automatu P_1 budú mať tvar $[q, \alpha]$, $q \in Q$, $\alpha \in \Gamma^*$, $0 \leqq |\alpha| \leqq k$. Reťazec α zodpovedá reťazcu na vrchu zásobníka automatu P; prípad $|\alpha| = k$ dáva k dispozícii informáciu o k symboloch na vrchu zásobníka automatu P. Automat P_1 bude sledovať činnosť automatu P tak, že ak P nahradzuje m vrchných symbolov zásobníka reťazcom dĺžky n, tak automat P_1 vykoná túto akciu tým, že prejde do vnútorného stavu, v ktorom je rovnakým spôsobom zmenená jeho

druhá zložka. Ak $m > n$, tak P_1 realizuje $(m - n)$ e-prechodov, ktorým sa odoberie $(m - n)$ vrchných symbolov zásobníka a aktualizuje sa stav tak, aby sa zachovala informácia o k vrchných symboloch zásobníka. Ak je $m < n$, tak sa rovnakým mechanizmom aktualizuje stav i vrch zásobníka (presunom stavovej informácie na vrch zásobníka).

Formálne môžeme konštrukciu zásobníkového automatu P_1 opísať takto:

1. $Q_1 = \{[q, \alpha]/q \in Q, \alpha \in \Gamma^*, 0 \leqq |\alpha| \leqq k\}$.
2. $\Gamma_1 = \Gamma \cup \{Z_1\}$.
3. Nech $\delta(q, a, X_1 \dots X_m)$ obsahuje prvok $(r, Y_1 \dots Y_n)$;
   a) ak $m \geqq n$, tak pre všetky $Z \in \Gamma_1$ také, že $|\alpha| = k - m$, obsahuje $\delta_1([q, X_1 \dots X_m\alpha], a, Z)$ prvok $([r, Y_1 \dots Y_n\alpha], e)$,
   b) ak $m < n$, tak pre všetky $Z \in \Gamma_1$, $\alpha \in \Gamma_1^*$ také, že $|\alpha| = k - m$ obsahuje $\delta_1([q, X_1 \dots X_m\alpha], a, Z)$ prvok $([r, \beta], \gamma Z)$, pričom $\beta\gamma = Y_1 \dots Y_n\alpha$ a $|\beta| = k$
   c) pre všetky $q \in Q$, $Z \in \Gamma_1$, $\alpha \in \Gamma_1^*$ pre ktoré $|\alpha| < k$ je $\delta_1([q, \alpha], e, Z) = \{([q, \alpha Z], e)\}$.
4. $q_1 = [q_0, Z_0Z_1^{k-1}]$.
5. $F_1 = \{[q, \alpha]/q \in F, \alpha \in \Gamma_1^*\}$.

Indukciou možno dokázať, že $L(P_1) = l(P)$.

Druhá modifikácia definície zásobníkového automatu sa dotýka jeho koncovej konfigurácie (q, e, γ), $q \in F$, $\gamma \in \Gamma^*$. Podmienka, že je prečítaný celý vstupný reťazec, zrejme nie je postačujúca, pretože by automat P s každou vetou $w \in L(P)$ prijímal všetky jej predpony. Preto je súčasťou definície automatu množina koncových stavov, z ktorej sa odvodzuje druhá podmienka pre koncovú konfiguráciu. Na tretiu zložku konfigurácie, obsah zásobníka, sa nekládla žiadna podmienka.

Teraz si ukážeme, že práve túto tretiu zložku konfigurácie môžeme využiť namiesto koncového stavu. Reťazec w bude zásobníkovým automatom prijatý práve vtedy, keď bude prečítaný daný reťazec a obsah zásobníka sa vyprázdni. Koncová konfigurácia bude mať teda tvar (q, e, e), pričom stav q môže byť teraz ľubovoľný. Ak platí $(q_0, w, Z_0) \overset{*}{\vdash}_P (q, e, e)$, tak hovoríme, že w je prijatý *vyprázdnením zásobníka*. Automat P, ktorý prijíma reťazec vyprázdnením zásobníka a špecifikuje jazyk

$$L(P) = \{w/(q_0, w, Z_0) \overset{*}{\vdash} (q, e, e)\}$$

budeme označovať ako sedmic.

$$P = (Q, T, \Gamma, \delta, q_0, Z_0, \emptyset)$$

t. j. namiesto teraz nefunkčnej rnnožiny koncových stavov zapisujeme konvenčne symbol prázdnej množiny.

Veta 5.4. Nech $P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Potom existuje taký zásobníkový automat P_1 prijímajúci reťazce vyprázdnením zásobníka, že $L(P) = L(P_1)$.

Dôkaz. Automat P_1 budeme znova konštruovať tak, aby sledoval činnosť automatu P. Kedykoľvek automat P dospeje do koncového stavu, bude automat P_1 alternatívne buď pokračovať v činnosti automatu P, alebo prejde do špeciálneho stavu q_e, ktorý spôsobí vyprázdnenie zásobníka. Musíme však uvážiť situáciu, keď automat P je v konfigurácii s prázdnym zásobníkom, nie však v koncovovm stave. Aby sme zabránili prípadom, že automat P_1 prijíma reťazec, ktorý nemá byť prijatý, pridáme k zásobníkovej abecede automatu P_1 znak, ktorý bude označovať dno zásobníka a môže byť vybratý iba vtedy, ak je automat P_1 v stave q_e. Formálne, nech

$$P_1 = (Q \cup \{q_e, q_1\}, T, \Gamma \cup \{Z_1\}, \delta_1, q_1, Z_1, \emptyset)$$

Zobrazenie δ_1 teraz definujeme takto:

1. Ak $\delta(q, a, Z)$ obsahuje (r, γ), tak $\delta_1(q, a, Z)$ obsahuje (r, γ) pre všetky $q \in Q$, $a \in T \cup \{e\}$ a $Z \in \Gamma$.
2. $\delta_1(q_1, e, Z_0) = \{(q_0, Z_0Z_1)\}$. Prvý prechod zásobníkového automatu P_1 uloží do zásobníka reťazec Z_0Z_1, kde Z_1 je špeciálny znak označujúci dno zásobníka, a prejde do začiatočného stavu automatu P.
3. Pre všetky $q \in F$ a $Z \in \Gamma \cup \{Z_1\}$ obsahuje $\delta_1(q, e, Z)$ prvok (q_e, e).
4. Pre všetky $Z \in \Gamma \cup \{Z_1\}$ sa $\delta(q_e, e, Z) = \{(q_e, e)\}$.

Vlastný dôkaz, že $L(P) = L(P_1)$, je jednoduchý a ponecháme ho ako cvičenie.

Veta 5.4 platí aj obrátene. K zásobníkovému automatu P prijímajúcemu vstupný reťazec vyprázdnením zásobníka možno zostrojiť zásobníkový automat P_1 prijímajúci jazyk $L(P)$ prechodom do koncového stavu. Automat P_1 získame zavedením koncového stavu q_F a takou modifikáciou prechodovej funkcie, ktorá na začiatku uloží na dno zásobníka špeciálny symbol Z' a ďalej realizuje prechod

$$(q, e, Z') \vdash (q_F, e, e)$$

pre ľubovoľný stav $q \in Q$.

Teraz už môžeme prejsť k hlavnej vete tohto článku.

Veta 5.5. Nech P je zásobníkový automat. Potom existuje bezkontextová gramatika G, pre ktorú platí $L(G) = L(P)$.

Dôkaz. Na základe viet 5.3 a 5.4 môžeme bez ujmy na všeobecnosti predpokladať, že automat

$$P = (Q, T, \Gamma, \delta, q_0, Z_0, \emptyset)$$

prijíma reťazce vyprázdnením zásobníka a že prechodová funkcia má tvar

$\delta: Q \times (T \cup \{e\}) \times \Gamma \to 2^{Q \times \Gamma^*}$

Vlastná konštrukcia gramatiky G sa opiera o neterminálne symboly tvaru $[qZp]$, $q, p \in Q$, $Z \in \Gamma$, a o analógiu postupnosti prechodov automatu P a ľavej derivácie v gramatike G. Presnejšie, symboly každej vetnej formy tvoriace ľavú deriváciu vety w budú korešpondovať s už spracovanými symbolmi zo vstupnej pásky (terminálnou predponou ľavej vetnej formy) a so symbolmi v zásobníku.

Formálne bude gramatika G definovaná takto:

$G = (N, T, P, S)$

kde 1. $N = \{S\} \cup \{[qZp]/q, p \in Q, Z \in \Gamma\}$

2. Množina P má prepisovacie pravidlá tvaru:

a) $S \to [q_0Z_0q]$ pre všetky $q \in Q$,

b) $[qAp] \to a[q_1B_1q_2][q_2B_2q_3] \ldots [q_mB_mq_{m+1}]$ pre všetky $q_1, q_2, \ldots, q_{m+1}$ z Q, kde $q_{m+1} = p$, a pre všetky $a \in T \cup \{e\}$ a $A, B_1, B_2, \ldots, B_m$ z Γ také, že $\delta(q, a, A)$ obsahuje $(q_1, B_1B_2 \ldots B_m)$. Ak $m = 0$, tak $q_1 = p$, $\delta(q, a, A)$ obsahuje (p, e) a zodpovedajúce pravidlo má tvar $[qAp] \to a$.

Možno dokázať, že platí

$(q_0, w, Z_0) \vdash_P^* (q, e, e)$ práve vtedy, ak $S \underset{G}{\Rightarrow} [q_0Z_0q] \underset{G}{\overset{*}{\Rightarrow}} w$, a teda $L(P) = L(G)$.

Príklad 5.6. Máme zásobníkový automat P

$P = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$

kde zobrazenie δ je dané takto:

$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$ $\quad \delta(q_1, 1, X) = \{(q_1, e)\}$
$\delta(q_0, 0, X) = \{(q_0, XX)\}$ $\quad \delta(q_1, e, X) = \{(q_1, e)\}$
$\delta(q_0, 1, X) = \{(q_1, e)\}$ $\quad \delta(q_1, e, Z_0) = \{(q_1, e)\}$

Vytvoríme gramatiku $G = (N, T, P, S)$ takú, že $L(P) = L(G)$. Množiny neterminálnych a terminálnych symbolov majú tvar:

$N = \{S, [q_0Xq_0], [q_0Xq_1], [q_1Xq_0], [q_1Xq_1],$
$[q_0Z_0q_0], [q_0Z_0q_1], [q_1Z_0q_0], [q_1Z_0q_1]\}$
$T = \{0, 1\}$

Množina N obsahuje niektoré neterminály, ktoré sú v gramatike G nedostupné. Aby sme ich nemuseli dodatočne odstraňovať, začneme vytvárať pravidlá gramatiky G začínajúc S-pravidlami a budeme pridávať iba tie neterminály, ktoré sa objavujú na pravých stranách získaných pravidiel. Podľa bodu *a)* vytvoríme S-pravidlá:

$S \to [q_0Z_0q_0]$ $\quad S \to [q_0Z_0q_1]$

Teraz pridáme pravidlá pre neterminál $[q_0Z_0q_0]$

$[q_0Z_0q_0] \to 0[q_0Xq_0][q_0Z_0q_0]$
$[q_0Z_0q_0] \to 0[q_0Xq_1][q_1Z_0q_0]$

vyplývajúce z $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$.

Pravidlá pre neterminál $[q_0Z_0q_1]$

$[q_0Z_0q_1] \to 0[q_0Xq_0][q_0Z_0q_1]$
$[q_0Z_0q_1] \to 0[q_0Xq_1][q_1Z_0q_1]$

sú požadované zobrazením $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$.

Pravidlá pre zostávajúce neterminály sú

$[q_0Xq_0] \to 0[q_0Xq_0][q_0Xq_0]$
$[q_0Xq_0] \to 0[q_0Xq_1][q_1Xq_0]$
$[q_0Xq_1] \to 0[q_0Xq_0][q_0Xq_1]$
$[q_1Xq_1] \to 0[q_0Xq_1][q_1Xq_1]$

pretože $\delta(q_0, 0, X) = \{(q_0, XX)\}$;

$[q_0Xq_1] \to 1$, pretože $\delta(q_0, 1, X) = \{(q_1, e)\}$
$[q_1Z_0q_1] \to e$, pretože $\delta(q_1, e, Z_0) = \{(q_1, e)\}$
$[q_1Xq_1] \to e$, pretože $\delta(q_1, e, X) = \{(q_1, e)\}$
$[q_1Xq_1] \to 1$, pretože $\delta(q_1, 1, X) = \{(q_1, e)\}$

Všimnime si teraz, že pre neterminály $[q_1Xq_0]$ a $[q_1Z_0q_0]$ nie sú definované žiadne pravidlá, a preto ani z neterminálu $[q_0Z_0q_0]$, ani z $[q_0Xq_0]$ nemožno derivovať terminálne reťazce. Ak teda odstránime pravidlá obsahujúce niektorý z uvedených štyroch neterminálov, dostaneme ekvivalentnú gramatiku generujúcu jazyk $L(P)$

$S \to [q_0Z_0q_1]$
$[q_0Z_0q_1] \to 0[q_0Xq_1][q_1Z_0q_1]$
$[q_0Xq_1] \to 0[q_0Xq_1][q_1Xq_1]$
$[q_1Z_0q_1] \to e$
$[q_0Xq_1] \to 1$
$[q_1Xq_1] \to 1 \mid e$

Napríklad postupnosti prechodov

$(q_0, 01, Z_0) \vdash (q_0, 1, XZ_0) \vdash (q_1, e, Z_0) \vdash (q_1, e, e)$

automatu P, zodpovedá v G táto ľavá derivácia reťazca 01:

$S \Rightarrow [q_0Z_0q_1] \Rightarrow 0[q_0Xq_1][q_1Z_0q_1] \Rightarrow 01\,[q_1Z_0q_1] \Rightarrow 01$

Na záver zhrňme výsledky viet 5.1 a 5.2 a výsledky tohto článku.

Ukázali sme, že bezkontextové jazyky a jazyky prijímané zásobníkovými automatmi tvoria rovnakú triedu jazykov. Každý jazyk tejto triedy možno špecifikovať:

— zásobníkovým automatom, ktorý prijíma vstupné reťazce prechodom do koncového stavu,
— zásobníkovým automatom, ktorý prijíma vstupné reťazce vyprázdnením zásobníka,
— zásobníkovým automatom, ktorý sa rozhoduje podľa symbolu na vrchu zásobníka,
— zásobníkovým automatom, ktorý sa rozhoduje podľa reťazca na vrchu zásobníka.

5.4 DETERMINISTICKÉ ZÁSOBNÍKOVÉ AUTOMATY

Už v úvode tejto kapitoly sme diskutovali o zdrojoch nedeterminizmu všeobecného zásobníkového automatu. Teraz definujeme triedu zásobníkových automatov, ktoré môžu v každom kroku prechádzať iba do jedinej konfigurácie — deterministické zásobníkové automaty.

Definícia 5.5. Zásobníkový automat $P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ nazývame *deterministický zásobníkový automat*, ak spĺňa tieto podmienky:

1. $|\delta(q, a, \alpha)| \leqq 1$ pre všetky $q \in Q$, $a \in T \cup \{e\}$, $\alpha \in \Gamma^*$.
2. Pre ľubovoľný stav q, symbol $a \in T \cup \{e\}$, a reťazce α, β zásobníkových symbolov platí:
   a) ak $\delta(q, a, \alpha) \neq \emptyset$ a zároveň $\delta(q, e, \beta) \neq \emptyset$, tak α nie je predponou reťazca β, ani β nie je predponou reťazca α,
   b) ak $\delta(q, a, \alpha) \neq \emptyset$ a zároveň $\delta(q, a, \beta) \neq \emptyset$, α nie je predponou reťazca β, ani β nie je predponou reťazca α.
   (Prázdny reťazec je predponou každého reťazca.)

P r í k l a d 5.7. Deterministický zásobníkový automat P, ktorý prijíma jazyk $L = \{wcw^R / w \in \{a, b\}^+\}$ možno definovať takto:

$P = (\{q_0, q_1\}, \{a, b, c\}, \{Z, a, b\}, \delta, q_0, Z, \emptyset)$
$\delta(q_0, X, e) = (q_0, X)$ pre všetky $X \in \{a, b\}$
$\delta(q_0, c, e) = (q_1, e)$
$\delta(q_1, X, X) = (q_1, e)$ pre všetky $X \in \{a, b\}$
$\delta(q_1, e, Z) = (q_1, e)$

Automat P pracuje tak, že najskôr presunie všetky vstupné symboly až po stredový symbol c do zásobníka. Po prečítaní symbolu c prejde do stavu q_1 a teraz skúma, či sa zhoduje čítaný symbol so symbolom na vrchole zásobníka. Vstupný reťazec je prijímaný vyprázdnením zásobníka.

Pokiaľ pracujeme so zásobníkovým automatom, ktorý sa rozhoduje iba podľa vrcholu zásobníka, potom sú podmienky pre determinizmus prechodov jednoduchšie a majú tvar:

1. Prechodová funkcia δ spĺňa podmienku $|\delta(q, a, \alpha)| \leqq 1$ pre všetky $q \in Q$, $a \in T \cup \{e\}$.
2. Pre ľubovoľný stav q, vstupný symbol a a zásobníkový symbol Z neplatí súčasne $\delta(q, a, Z) \neq \emptyset$ a $\delta(q, e, Z) \neq \emptyset$.

V predchádzajúcom článku sme ukázali, že zásobníkové automaty prijímajú práve bezkontextové jazyky. Vzniká teda otázka, či ľubovoľný bezkontextový jazyk môže byť prijímaný i deterministickým zásobníkovým automatom alebo inak povedané, či možno každý zásobníkový automat upraviť na ekvivalentný deterministický zásobníkový automat podobne, ako je to v triede konečných automatov. Odpoveď na túto otázku je záporná. Jazyky prijímané deterministickými zásobníkovými automatmi tvoria vlastnú podtriedu bezkontextových jazykov.

Definícia 5.6. Jazyk L sa nazýva *deterministický bezkontextový jazyk*, ak existuje deterministický zásobníkový automat P taký, že $L(P) = L$.

Veta 5.6. Existujú bezkontextové jazyky, ktoré nie sú deterministickými bezkontextovými jazykmi.

Dôkaz. Veta 5.6 je dôsledkom niektorých uzáverových vlastností deterministických bezkontextových jazykov, ktoré tu však neuvádzame. K najznámejším patrí uzavretosť vzhľadom na doplnok, ktorá vyplýva zo skutočnosti, že ku každému deterministickému zásobníkovému automatu P možno vytvoriť deterministický zásobníkový automat P' taký, že platí $L(P') = \overline{L(P)}$.

Predpokladajme teraz, že každý bezkontextový jazyk L môžeme prijímať deterministickým zásobníkovým automatom. Potom môžeme deterministickým zásobníkovým automatom pripínať tiež jazyk $\bar{L}$. Ale bezkontextové jazyky nie sú uzavreté vzhľadom na doplnok (veta 4.13), a teda $\bar{L}$ nemusí byť bezkontextový jazyk, čo je v spore s predpokladom.

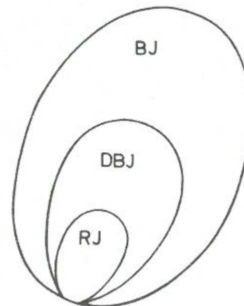
S príkladom bezkontextového jazyka, ktorý nie je deterministickým bezkontextovým jazykom, sme sa stretli v príklade 5.3. Intuitívne cítime, že napriek tomu, že deterministický zásobníkový automat disponuje nekonečnou zásobníkovou pamäťou, konečná množina vnútorných stavov a špecifický prístup do zásobníkovej pamäti nedávajú možnosť uchovávať informáciu o ľubovoľnom reťazci nekonečného jazyka $L = \{ww^R / w \in T^*\}$ takým spôsobom, aby jeho prijatie prebiehalo deterministicky.

Deterministické bezkontextové jazyky zahrňujúce, pochopiteľne, všetky regulárne jazyky tvoria veľmi dôležitú triedu formálnych jazykov (*obr. 5.2*).

K najvýznamnejším vlastnostiam deterministických bezkontextových jazykov z hľadiska ich aplikácií v prekladačoch programovacích jazykov patria

— efektívnosť syntaktickej analýzy,
— automatizovateľnosť výstavby syntaktického analyzátora,
— jednoduchosť lokalizácie syntaktických chýb.

V ďalších článkoch sa budeme zaoberať dvoma najdôležitejšími triedami deterministických bezkontextových jazykov.



Obr. 5.2. Hierarchia bezkontextových jazykov. BJ — bezkontextové jazyky, DBJ — deterministické bezkontextové jazyky, RJ — regulárne jazyky

Cvičenia

1. Vytvorte zásobníkový automat, ktorý prijíma jazyk

$$L = \{a^n b^m / n \leqq m \leqq 2n\}$$

2. Pre gramatiky

a) $S \to aSb \mid e$
b) $S \to AS \mid b$
$A \to SA \mid a$
c) $S \to SS \mid A$
$A \to 0A1 \mid S \mid 01$

vytvorte zásobníkové automaty modelujúce syntaktickú analýzu zhora nadol a zdola nahor.

3. Podľa vety 5.3 vytvorte k zásobníkovému automatu z príkladu 5.3 ekvivalentný zásobníkový automat, ktorý sa rozhoduje iba podľa vrchu zásobníka.

4. Vyjadrite formálne konštrukciu ekvivalentného zásobníkového automatu, ktorý prijíma vstupné reťazce prechodom do koncového stavu, k zásobníkovému automatu prijímajúcemu vstupné reťazce vyprázdnením zásobníka.

5. Nájdite gramatiku, ktorá generuje jazyk $L(P)$, kde

$$P = (\{q_0, q_1, q_2\}, \{a, b\}, \{Z_0, A\}, \delta, q_0, Z_0, \{q_2\})$$

a zobrazenie δ je definované takto:

$$\delta(q_0, a, Z_0) = \{(q_1, Az_0)\}$$
$$\delta(q_0, a, A) = \{(q_1, AA)\}$$
$$\delta(q_1, a, A) = \{(q_0, AA)\}$$
$$\delta(q_1, e, A) = \{(q_2, A)\}$$
$$\delta(q_2, b, A) = \{(q_2, e)\}$$

6. Ukážte, že jazyky L_1 a L_2,

$L_1 = \{c0^n1^n / n \geqq 1\} \cup \{0^n1^{2n} / n \geqq 1\}$

$L_2 = \{w / w \in \{a, b\}^*, w \text{ obsahuje rovnaký počet } a \text{ aj } b\}$

sú deterministické bezkontextové jazyky.

6 SYNTAKTICKÁ ANALÝZA DETERMINISTICKÝCH BEZKONTEXTOVÝCH JAZYKOV

6.1 LL GRAMATIKY A JAZYKY

V tejto kapitole sa budeme zaoberať skupinou algoritmov syntaktickej analýzy, ktoré vytvárajú derivačný strom analyzovaného reťazca smerom zhora nadol. Tieto algoritmy sa nazývajú *algoritmy LL(k) analýzy*, pretože čítajú vstupný reťazec zľava doprava, vytvárajú ľavý rozklad, a pritom používajú informáciu o najbližších k symboloch z doteraz neprečítanej časti vstupného reťazca. Gramatiky, pre ktoré je možné tieto algoritmy vytvoriť, sa nazývajú *LL(k) gramatiky*. Okrem všeobecných LL(k) gramatík sa budeme zaoberať špeciálnymi prípadmi LL(k) gramatík: *jednoduchými LL(1) gramatikami*, *q-gramatikami*, *LL(1) gramatikami* a *silnými LL(k) gramatikami*. Spoločne ich budeme nazývať *LL gramatiky*. Základný princíp syntaktickej analýzy pre LL gramatiky môžeme formulovať takto:

Daná je bezkontextová gramatika $G = (N, T, P, S)$ a reťazec $w = a_1a_2 \dots a_n$, ktorý je vetou jazyka $L(G)$. Potom existuje ľavá derivácia $S = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_m = w$. Vzhľadom na to, že derivácia je ľavá, má každá vetná forma γ_i tvar $\gamma_i = a_1a_2 \dots a_jA_i\beta_i$, kde $a_1, a_2, \dots, a_j$ sú terminálne symboly, A je neterminálny symbol, β je reťazec terminálnych a neterminálnych symbolov. Pritom reťazec $a_1a_2 \dots a_j$ je predponou vety w.

Predpokladajme, že $A \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_p$ sú všetky pravidlá z množiny pravidiel P s neterminálnym symbolom A na ľavej strane. Potom základný problém syntaktickej analýzy metódou zhora nadol spočíva v nájdení toho pravidla $A \to \alpha_k$, ktorého aplikáciou dostaneme z vetnej formy γ_i vetnú formu γ_{i+1}.

Modelom syntaktického analyzátora pracujúceho metódou zhora nadol je zásobníkový automat vytvorený v kapitole 5. Tento zásobníkový automat je nedeterministický, a preto ho nemožno priamo použiť ako syntaktický analyzátor.

Vráťme sa k jeho vytváraniu a rozanalyzujme, za akých okolností a akým spôsobom je možné vytvoriť deterministický zásobníkový automat.

Pre bezkontextovú gramatiku pomocou konštrukcie z kapitoly 5 vytvoríme zásobníkový automat, ktorého zobrazenie δ je definované takto:

1. $\delta(q, e, A) = \{(q, \alpha) / A \to \alpha \in P\}$.
2. $\delta(q, a, a) = \{(q, e)\}$.

Operáciu podľa bodu 1 budeme nazývať *expanzia*, operáciu podľa bodu 2 *porovnanie*.

Zobrazenie podľa bodu 2 vyhovuje definícii deterministického zásobníkového automatu. Problémom je zobrazenie podľa bodu 1. Zobrazenie $\delta(q, e, A)$ dáva množinu, ktorá má toľko prvkov, koľko pravidiel pre neterminálny symbol A v gramatike existuje. Je teda vidieť, že problém výberu pravidla táto konštrukcia nerieši. Deterministický zásobníkový automat vznikne touto konštrukciou iba vtedy, keď pre každý neterminálny symbol je v gramatike iba jedno pravidlo. Tieto gramatiky (LL(0) gramatiky) nemajú praktický význam z toho dôvodu, že generujú iba jedinú vetu alebo prázdny jazyk.

Pre väčšinu bezkontextových gramatík vznikne teda uvedenou konštrukciou nedeterministický zásobníkový automat. Pritom si všimnime, že pre výber pravidla na expanziu tento automat nevyužíva žiadnu informáciu. Pri syntaktickej analýze existujú dva zdroje informácií, ktoré možno používať pri rozhodovaní o ďalšom postupe:

1. Informácie o doteraz neprečítanej časti vstupného reťazca.
2. Informácie o doterajšom priebehu analýzy.

Pri syntaktickej analýze metódou zhora nadol sa využíva predovšetkým informácia o k najbližších symboloch v doteraz neprečítanej časti vstupného reťazca. To znamená, že podľa tohto vopred prezeraného reťazca sa vykonáva rozhodnutie o tom, aké pravidlo použiť pri expanzii. Informácia o doterajšom priebehu analýzy sa uplatňuje iba pri LL(k) gramatikách, ktoré nie sú silné LL(k) gramatiky.

V nasledujúcich článkoch sa budeme zaoberať jednotlivými triedami LL gramatík. Odsek 6.1.1 o silných LL gramatikách rozoberá jednoduché LL(1) gramatiky, q-gramatiky, LL(1) gramatiky a silné LL(k) gramatiky. V odseku 6.1.2 sa budeme zaoberať všeobecnými LL(k) gramatikami, pre ktoré je pri syntaktickej analýze potrebné okrem informácie o vopred prezeranom reťazci využívať aj informáciu o doterajšom priebehu syntaktickej analýzy.

Pre všetky triedy LL gramatík sa používa okrem drobných modifikácií ten istý algoritmus syntaktickej analýzy, založený na princípe zásobníkového automatu. Pri rozhodnutí o tom, akú expanziu v danom okamihu vykonať, sa vo všetkých prípadoch používa rozkladová tabuľka, v ktorej sú uvedené potrebné informácie. Táto tabuľka sa vytvára na základe danej bezkontextovej gramatiky. Spôsob vytvárania rozkladovej tabuľky je pre rôzne triedy LL gramatík odlišný a v ďalších odsekoch je opísaný pre jednotlivé prípady.

6.1.1 Silné LL gramatiky

6.1.1.1 Jednoduché LL(1) gramatiky

Jednoduché LL(1) gramatiky sú najjednoduchšou, a pritom už prakticky zaujímavou triedou LL gramatík. Princíp konštrukcie rozkladovej tabuľky najprv ukážeme na príklade.

Príklad 6.1. Daná je bezkontextová gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$, kde množina P obsahuje tieto pravidlá:

$S \to aASc$ $\quad$ $A \to a$

$S \to b$ $\quad$ $A \to cSAb$

Veta *acbabbc* má v gramatike G túto ľavú deriváciu:

$$S \Rightarrow aASc \Rightarrow acSAbSc \Rightarrow acbAbSc \Rightarrow acbaSc \Rightarrow acbabbc$$

Vidno, že v tejto gramatike môžeme deterministicky vytvoriť deriváciu každej vety, pretože výber pravej strany pravidla pre neterminálne symboly S a A je jednoznačne určený nasledujúcim symbolom vo vstupnom reťazci. To znamená, že vo vetnej forme $a_1a_2 \dots a_jX\alpha$ môžeme neterminálny symbol X nahradiť pravou stranou pravidla v závislosti od symbolu a_{j+1}.

Uvedieme teraz rozkladovú tabuľku, obsahujúcu pravé strany pravidiel, ktoré použijeme pre našu gramatiku na expanziu neterminálneho symbolu vo vetnej forme v závislosti od najbližšieho vstupného symbolu (*tab. 6.1*).

Tabuľka 6.1

	a	*b*	*c*
S	*aASc*	*b*	
A	*a*		*cSAb*

Tabuľku tohto tvaru budeme nazývať *rozkladová tabuľka*. Obsahuje informácie o tom, aké pravidlo použiť na expanziu neterminálneho symbolu, ktorý je na vrchu zásobníka, v závislosti od toho, aký symbol nasleduje vo vstupnom reťazci. Ak pre určitú kombináciu neterminálneho symbolu na vrchu zásobníka a vstupného symbolu nie je žiadná expanzia možná, ide o chybovú situáciu. Položku tabuľky, ktorá zodpovedá chybovej situácii, budeme vo všetkých príkladoch nechávať prázdnu. V našom príklade ide o položky pre kombinácie (S, c) a (A, b).

Teraz uvedieme definíciu jednoduchej LL(1) gramatiky, algoritmus na vytváranie rozkladovej tabuľky a nakoniec algoritmus syntaktickej analýzy. Pretože výstupom syntaktického analyzátora bude ľavý rozklad, pravidlá gramatiky očíslujeme a do rozkladovej tabuľky doplníme čísla pravidiel.

Definícia 6.1. Bezkontextová gramatika sa nazýva *jednoduchá LL*(1) *gramatika* (aj *S-gramatika*), keď platia nasledujúce dve podmienky:

1. Pravá strana každého pravidla začína terminálnym symbolom.
2. Ak dve pravidlá majú rovnakú ľavú stranu, ich pravé strany začínajú rôznymi terminálnymi symbolmi.

Pre každú jednoduchú LL(1) gramatiku môžeme vytvoriť rozkladovú tabuľku (označíme ju M), a to pomocou nasledujúceho algoritmu.

Algoritmus 6.1.
Vytvorenie rozkladovej tabuľky pre jednoduchú LL(1) gramatiku.
Vstup: jednoduchá LL(1) gramatika $G = (N, T, P, S)$.
Výstup: rozkladová tabuľka M pre gramatiku G.
Metóda: rozkladová tabuľka M je definovaná na karteziánskom súčine $N \times T$.
1. Ak $A \to a\alpha$ je i-té pravidlo v P, tak $M(A, a) = a\alpha, i$.
2. $M(X, a) =$ chyba vo všetkých ostatných prípadoch.

Príklad 6.2. V gramatike z príkladu 6.1 očíslujeme jednotlivé pravidlá takto:

1. $S \to aASc$ $\quad$ 3. $A \to a$
2. $S \to b$ $\quad$ 4. $A \to cSAb$

Aplikáciou algoritmu 6.1 na túto gramatiku získame rozkladovú tabuľku (*tab. 6.2*):

Tabuľka 6.2

M	*a*	*b*	*c*
S	*aASc*, 1	*b*, 2	
A	*a*, 3		*cSAb*, 4

Algoritmus 6.2.
Syntaktická analýza pre LL(1) gramatiky.
Vstup: rozkladová tabuľka M pre LL(1) gramatiku $G = (N, T, P, S)$, vstupný reťazec $w \in T^*$.
Výstup: ľavý rozklad vstupného reťazca v prípade, že $w \in L(G)$, inak chybová singalizácia.
Metóda: algoritmus číta vstupný reťazec, používa zásobník a vytvára výstupný reťazec. Konfigurácia algoritmu je trojica (x, α, π), kde $x \in T^*$ je doteraz neprečítaná časť vstupného reťazca, $\alpha \in (N \cup T)^*$ je obsah zásobníka a $\pi \in C^*$ je postupnosť čísel pravidiel použitých pri doteraz vykonaných expanziách. Na začiatku je v zásobníku symbol S. Zásobník má vrch vľavo. Výstupný

reťazec je prázdny. Algoritmus vykonáva prechody podľa 1 a 2, kým sa neobjaví situácia podľa 3 alebo 4.

1. Expanzia: $(ax, A\alpha, \pi) \vdash (ax, \beta\alpha, \pi i)$, ak $A \in N$ a $M(A, a) = \beta, i$. Symbol A na vrchu zásobníka je nahradený reťazcom β a číslo pravidla i je pripojené k výstupnej postupnosti.
2. Porovnanie: $(ax, a\alpha, \pi) \vdash (x, \alpha, \pi)$, ak $a \in T$. Symbol na vstupe sa porovnáva so symbolom v zásobníku a v prípade rovnosti sa vstupný symbol prečíta a zásobníkový symbol sa zo zásobníka vylúči.
3. Prijatie: v konfigurácii (e, e, π) analýza končí; π je ľavý rozklad vstupného reťazca.
4. Chyba: vo všetkých ostatných prípadoch analýza končí chybovou signalizáciou.

Poznámka 6.1. Algoritmus 6.2 možno použiť ako syntaktický analyzátor pre všetky LL(1) gramatiky (pozri ďalej), to znamená, že jeho použiteľnosť nie je obmedzená iba na jednoduché LL(1) gramatiky.

Na tento algoritmus sa môžeme pozerať ako na deterministický zásobníkový automat. Jeho princíp je založený na zásobníkovom automate vytvorenom v kapitole 5.

Výber pravidla pri expanzii sa vykonáva deterministicky pomocou rozkladovej tabuľky. Oproti zásobníkovému automatu sú pridané ďalšie dve operácie, a to prijatie a chyba. Operácia prijatie znamená ukončenie analýzy normálnym spôsobom a výsledkom je ľavý rozklad. Operácia chyba znamená ukončenie analýzy v okamihu, keď algoritmus zistí chybu vo vstupnom reťazci. Táto situácia môže nastať v týchto konfiguráciách:

$(ax, A\alpha, \pi)$ v prípade, že $M(A, a) =$ chyba, t. j. keď nie je možná expanzia,
$(ax, b\alpha, \pi)$, keď $b \in T$, $a \neq b$, t. j., keď nie je možné zrovnanie,
(x, e, π), keď zásobník je prázdny a vstupný reťazec nie je prečítaný,
(e, α, π), keď vstupný reťazec je celý prečítaný a zásobník nie je prázdny.

Posledný prípad je chybovou situáciou pre jednoduché LL(1) gramatiky. Pre ostatné typy LL gramatík táto situácia nemusí znamenať chybu.

Príklad 6.3. Použitím rozkladovej tabuľky z príkladu 6.2 vykonáme analýzu reťazca *acbabbc*. Algoritmus syntaktickej analýzy vykoná túto postupnosť prechodov:

$(acbabbc, S, e) \vdash (acbabbc, aASc, 1)$
$\vdash (cbabbc, ASc, 1)$
$\vdash (cbabbc, cSAbSc, 14)$
$\vdash (babbc, SAbSc, 14)$
$\vdash (babbc, bAbSc, 142)$
$\vdash (abbc, AbSc, 142)$
$\vdash (abbc, abSc, 1423)$

$\vdash (bbc, bSc, 1423)$
$\vdash (bc, Sc, 1423)$
$\vdash (bc, bc, 14232)$
$\vdash (c, c, 14232)$
$\vdash (e, e, 14232)$

Skutočnosť, že daná gramatika je jednoduchá LL(1) gramatika, poznáme veľmi jednoducho podľa tvaru pravidiel.

Príklad 6.4. Gramatika $G_1 = (\{S, T\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

1. $S \to aT$
2. $S \to Tbs$
3. $T \to bT$
4. $T \to ba$

nie je jednoduchá LL(1) gramatika, pretože pravá strana pravidla 2 nezačína terminálnym symbolom a pravidlá 3 a 4 majú pravé strany s rovnakým začiatočným symbolom. Pritom je možné veľmi jednoducho ukázať, že gramatika $G_2 = (\{S, R\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

1. $S \to abR$
2. $S \to bRbS$
3. $R \to a$
4. $R \to bR$

je jednoduchá LL(1) gramatika a platí $L(G_1) = L(G_2)$.

6.1.1.2 *q-gramatiky*

V predchádzajúcom článku sme uviedli metódu konštrukcie rozkladovej tabuľky pre jednoduché LL(1) gramatiky. S touto metódou však nevystačíme napr. pri konštrukcii rozkladovej tabuľky pre tie jazyky, ktoré obsahujú prázdne reťazce. Ide napr. o také jednoduché jazyky ako:

$L_1 = \{1^n / n \geqq 0\}$
$L_2 = \{0^n 1^n / n \geqq 0\}$
$L_3 = \{1^n 0^m / 0 < n \leqq m\}$

Všimnime si, že definícia jednoduchých LL(1) gramatík vylučuje možnosť existencie pravidiel s prázdnou pravou stranou (e-pravidiel).

Preto teraz uvedieme, akým spôsobom možno vytvoriť syntaktický analyzátor pre gramatiky s e-pravidlami.

Príklad 6.5. Je daná gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$, kde P obsahuje tieto pravidlá:

1. $S \to aAS$
2. $S \to b$
3. $A \to cAS$
4. $A \to e$

Táto gramatika nie je jednoduchá LL(1) gramatika, pretože pravá strana pravidla 4 nezačína terminálnym symbolom. Napriek tomu možno vytvoriť pre takúto gramatiku rozkladovú tabuľku a vykonávať syntaktickú analýzu pomocou algoritmu 6.2. Rozkladová tabuľka pre gramatiku, ktorá obsahuje e-pravidlá, bude rozšírená o jeden stĺpec označený symbolom e, t. j. prázdnym reťazcom. Dôvodom k tomuto rozšíreniu je možnosť vykonávania expanzie pomocou e-pravidiel i v prípade, že vstupný reťazec je už celý prečítaný. To znamená, že konfigurácia (e, α, π) nemusí nutne viesť k chybovej sinalizácii.

Príklad 6.6. Rozkladová tabuľka pre gramatiku G z príkladu 6.5 (*tab. 6.3*) bude mať tvar:

Tabuľka 6.3

M	a	b	c	e
S	aAS, 1	b, 2		
A	e, 4	e, 4	cAS, 3	

Rozklad vety $aacbb$ pomocou tejto rozkladovej tabuľky bude prebiehať takto:

$$\begin{array}{llrrr}
(aacbb,\ S,\ e) & \vdash (& aacbb, & aAS, & 1) \\
 & \vdash (& acbb, & AS, & 1) \\
 & \vdash (& acbb, & S, & 14) \\
 & \vdash (& acbb, & aAS, & 141) \\
 & \vdash (& cbb, & AS, & 141) \\
 & \vdash (& cbb, & cASS, & 1413) \\
 & \vdash (& bb, & ASS, & 1413) \\
 & \vdash (& bb, & SS, & 14134) \\
 & \vdash (& bb, & bS, & 141342) \\
 & \vdash (& b, & S, & 141342) \\
 & \vdash (& b, & b, & 1413422) \\
 & \vdash (& e, & e, & 1413422)
\end{array}$$

Aby sme mohli vytvoriť rozkladovú tabuľku pre q-gramatiku $G = (N, T, P, S)$, zavedieme funkciu FOLLOW (X), kde X je neterminálny symbol.

Definícia 6.2. Pre neterminálny symbol X v bezkontextovej gramatike $G = (N, T, P, S)$ platí:

$$\text{FOLLOW}\ (X) = \{a/S \overset{*}{\Rightarrow} \alpha X\beta,\ \beta \overset{*}{\Rightarrow} a\gamma,\ \gamma \in (N \cup T)^*\} \cup \{e/S \overset{*}{\Rightarrow} \alpha X\}$$

Hodnotou funkcie FOLLOW (X) je množina obsahujúca všetky terminálne symboly, ktoré môžu byť vo vetných formách za symbolom X. V prípade, že X sa vyskytne na konci niektorej vetnej formy, množina FOLLOW (X) obsahuje i prázdny reťazec.

Príklad 6.7. Vypočítame FOLLOW (A) v gramatike G z príkladu 6.5. Symbol A sa môže vyskytnúť v deriváciách typu

$$S \Rightarrow aAS \Rightarrow acASS \Rightarrow \ldots$$

Vo všetkých prípadoch sa za symbolom A vo vetnej forme vyskytuje vždy symbol S. Pretože zo symbolu S možno derivovať reťazce začínajúce symbolmi a alebo b, platí, že FOLLOW $(A) = \{a, b\}$.

Funkciu FOLLOW zavádzame preto, že symboly, ktoré sa vyskytujú na začiatku doteraz nespracovanej časti vstupného reťazca pri expanzii pomocou pravidla $X \rightarrow e$, sú práve tie, ktoré sú obsiahnuté vo FOLLOW (X).

Funkciu FOLLOW budeme využívať pri vytváraní rozkladovej tabuľky pre q-gramatiky.

Definícia 6.3. Bezkontextová gramatika sa nazýva *q-gramatika*, keď platí:

1. Pravá strana každého pravidla je buď prázdna, alebo začína terminálnym symbolom.
2. Ak dve pravidlá majú rovnakú ľavú stranu, tak ich pravé strany začínajú rôznymi terminálnymi symbolmi za predpokladu, že niektorá z nich nie je prázdny reťazec.
3. Ak sa v gramatike vyskytne pravidlo $A \rightarrow e$, tak musí platiť, že terminálne symboly, ktorými začínajú pravé strany ostatných pravidiel so symbolom A na ľavej strane, nesmú byť obsiahnuté vo FOLLOW (A).

Teraz uvedieme algoritmus na vytváranie rozkladovej tabuľky pre q-gramatiky.

Algoritmus 6.3.

Vytváranie rozkladovej tabuľky pre q-gramatiku.

Vstup: q-gramatika $G = (N, T, P, S)$.

Výstup: rozkladová tabuľka M pre gramatiku G.

Metóda: rozkladová tabuľka M je definovaná na karteziánskom súčine $N \times (T \cup \{e\})$.

1. Ak $A \rightarrow a\alpha$ je i-té pravidlo v P, tak $M(A, a) = a\alpha, i$.
2. Ak $A \rightarrow e$ je i-té pravidlo v P, tak $M(A, b) = e, i$ pre všetky $b \in$ FOLLOW (A).
3. $M(A, a) =$ chyba vo všetkých ostatných prípadoch.

Príklad 6.8. Je daná gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$, kde P obsahuje tieto pravidlá:

1. $S \rightarrow aA$	3. $A \rightarrow cSa$
2. $S \rightarrow b$	4. $A \rightarrow e$

Pre túto gramatiku vytvoríme rozkladovú tabuľku. Najprv zistíme, že FOLLOW $(A) = \{a, e\}$, pretože neterminálny symbol A sa môže vyskytnúť v deriváciách typu

$$S \Rightarrow aA \Rightarrow a$$
$$S \Rightarrow aA \Rightarrow acSa \Rightarrow acaAa \Rightarrow acaa$$

Pretože okrem pravidla $A \rightarrow e$ je pre neterminálny symbol A v gramatike iba pravidlo $A \rightarrow cSa$ a $c \notin$ FOLLOW$(A) = \{a, e\}$, je gramatika G q-gramatikou a môžeme pre konštrukciu rozkladovej tabuľky použiť algoritmus 6.3. Rozkladová tabuľka (*tab. 6.4*) bude mať tvar:

Tabuľka 6.4

M	a	b	c	e
S	aA, 1	b, 2		
A	e, 4		cSa, 3	e, 4

Ďalej urobíme analýzu reťazca $acaa$:

$$\begin{aligned}(acaa, S, e) &\vdash (\; acaa, \; aA, \; 1)\\ &\vdash (\; caa, \; A, \; 1)\\ &\vdash (\; caa, \; cSa, \; 13)\\ &\vdash (\; aa, \; Sa, \; 13)\\ &\vdash (\; aa, \; aAa, \; 131)\\ &\vdash (\; a, \; Aa, \; 131)\\ &\vdash (\; a, \; a, \; 1314)\\ &\vdash (\; e, \; e, \; 1314)\end{aligned}$$

6.1.1.3 LL(1) gramatiky

V predchádzajúcich dvoch pododsekoch sme sa zaoberali metódami konštrukcie rozkladových tabuliek pre jednoduché LL(1) gramatiky a q-gramatiky. Uvedené metódy možno aplikovať na gramatiky, v ktorých pravé strany pravidiel začínajú terminálnymi symbolmi, alebo ich tvoria prázdne reťazce. Teraz uvedené metódy zovšeobecníme na bezkontextové gramatiky, ktoré obsahujú pravidlá ľubovoľného tvaru. Pritom algoritmus syntaktickej analýzy zostane rovnaký a zmení sa iba algoritmus na vytváranie rozkladovej tabuľky.

Príklad 6.9. Daná je gramatika
$G = (\{E, E', T\}, \{a, +, (,)\}, P, E)$, kde P obsahuje pravidlá:

1. $E \rightarrow TE'$
2. $E' \rightarrow +TE'$
3. $E' \rightarrow e$
4. $T \rightarrow a$
5. $T \rightarrow (E)$

Táto gramatika nie je ani jednoduchá gramatika, ani q-gramatika, pretože pravá strana prvého pravidla začína neterminálnym symbolom. Napriek tomu možno pre túto gramatiku vytvoriť rozkladovú tabuľku. Táto tabuľka (*tab. 6.5*) má tvar:

Tabuľka 6.5

M	a	$+$	(	)	e
E	TE', 1		TE', 1		
E'		$+TE'$, 2		e, 3	e, 3
T	a, 4		(E), 5		

Aby sme mohli definovať LL(1) gramatiku, zavedieme, funkciu FIRST.

Definícia 6.4. Ak je daná gramatika $G = (N, T, P, S)$, tak pre $\alpha \in (N \cup T)^*$ platí:

$$\text{FIRST}(\alpha) = \{a / \alpha \Rightarrow^* a\beta,\ a \in T,\ \beta \in (N \cup T)^*\} \cup \{e / \alpha \stackrel{*}{\Rightarrow} e\}$$

Hodnotou funkcie FIRST(α) je množina terminálnych symbolov, ktoré sa môžu vyskytnúť na začiatku reťazcov derivovaných z α. Ak sa z α dá derivovať prázdny reťazec, tak ho obsahuje aj FIRST(α).

Pre gramatiku z príkladu 6.9 platí, že FIRST$(TE') = \{a, (\}$. To znamená, že z reťazca TE' možno derivovať reťazce začínajúce symbolmi a alebo (. Ako príklad možno uviesť derivácie:

$$TE' \Rightarrow aE' \Rightarrow a + TE' \Rightarrow a + aE' \Rightarrow a + a$$
$$TE' \Rightarrow (E)E' \Rightarrow (TE')E' \Rightarrow (aE')E' \Rightarrow (a + TE')E' \Rightarrow$$
$$(a + aE')E' \Rightarrow (a + a)E' \Rightarrow (a + a)$$

Preto pri expanzii symbolu E na vrchu zásobníka pri čítaní symbolu a alebo symbolu (vykonáme expanziu pomocou pravidla $E \rightarrow TE'$.

Definícia 6.5. Bezkontextová gramatika $G = (N, T, P, S)$ sa nazýva *LL*(1) *gramatika*, keď platí nasledujúca podmienka pre každé $A \in N$: Ak $A \rightarrow \alpha$ a $A \rightarrow \beta$ sú rôzne pravidlá v P, tak

a) pre reťazce α, β musí platiť

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$$

b) ak z reťazca α možno derivovať prázdny reťazec a z reťazca β nemožno derivovať prázdny reťazec, tak musí platiť

$$\text{FOLLOW}(A) \cap \text{FIRST}(\beta) = \emptyset$$

Poznámka 6.2. Predchádzajúcu definíciu možno skrátene formulovať takto: Bezkontextová gramatika $G = (N, T, P, S)$ sa nazýva LL(1) gramatika,

keď pre každé $A \in N$ platí podmienka: Ak $A \to \alpha$ a $A \to \beta$ sú rôzne pravidlá v P, tak

$$\mathrm{FIRST}(\alpha\,\mathrm{FOLLOW}(A)) \cap \mathrm{FIRST}(\beta\,\mathrm{FOLLOW}(A)) = \emptyset$$

V tomto prípade musíme rozšíriť definíciu funkcií FIRST a FOLLOW tak, aby ich argumenty mohli byť množiny reťazcov.

Definícia 6.6. Ak $G = (N, T, P, S)$ je bezkontextová gramatika a $R \subset (N \cup T)^*$ tak

$$\mathrm{FIRST}(R) = \{a / a \in \mathrm{FIRST}(\alpha) \text{ pre nejaké } \alpha \in R\}$$
$$\mathrm{FOLLOW}(R) = \{a / a \in \mathrm{FOLLOW}(\alpha) \text{ pre nejaké } \alpha \in R\}$$

Teraz sformulujeme algoritmus na vytváranie rozkladovej tabuľky pre LL(1) gramatiku.

Algoritmus 6.4.

Vytváranie rozkladovej tabuľky pre LL(1) gramatiku.

Vstup: LL(1) gramatika $G = (N, T, P, S)$.

Výstup: rozkladová tabuľka M pre gramatiku G.

Metóda: rozkladová tabuľka M je definovaná na karteziánskom súčine $N \times (T \cup \{e\})$.

1. Ak $A \to \alpha$ je i-té pravidlo v P, tak $M(A, a) = \alpha, i$ pre všetky $a \in \mathrm{FIRST}(\alpha) - \{e\}$.
2. Ak $A \to \alpha$ je i-té pravidlo v P a $e \in \mathrm{FIRST}(\alpha)$, tak $M(A, b) = \alpha, i$ pre všetky $b \in \mathrm{FOLLOW}(A)$.
3. $M(A, a)$ = chyba vo všetkých ostatných prípadoch.

Príklad 6.10. Vytvoríme rozkladovú tabuľku pre gramatiku $G = (\{E, E', T, T', F\}, \{a, +, *, [,]\}, P, E)$, kde P obsahuje pravidlá:

1. $E \to TE'$	4. $T \to FT'$	7. $F \to [E]$
2. $E' \to +TE'$	5. $T' \to *FT'$	8. $F \to a$
3. $E' \to e$	6. $T' \to e$	

Pri konštrukcii rozkladovej tabuľky použijeme tieto množiny:

$\mathrm{FIRST}(TE') = \{a, [\}$	$\mathrm{FIRST}(+TE') = \{+\}$
$\mathrm{FIRST}(FT') = \{a, [\}$	$\mathrm{FIRST}(*FT') = \{*\}$
$\mathrm{FOLLOW}(E') = \{e,]\}$	$\mathrm{FIRST}([E]) = \{[\}$
$\mathrm{FOLLOW}(T') = \{e,], +\}$	$\mathrm{FIRST}(a) = \{a\}$

Rozkladová tabuľka M (*tab. 6.6*) má tvar:

Tabuľka 6.6

M	a	$+$	$*$	$[$	$]$	e
E	$TE', 1$			$TE', 1$		
E'		$+TE', 2$			$e, 3$	$e, 3$
T	$FT', 4$			$FT', 4$		
T'		$e, 6$	$*FT', 5$		$e, 6$	$e, 6$
F	$a, 8$			$[E], 7$		

Ďalej urobíme analýzu reťazca $a + a$:

$$
\begin{aligned}
(a + a, E, e) &\vdash (\ a + a, \quad TE', \quad 1) \\
&\vdash (\ a + a, \quad FT'E', \quad 14) \\
&\vdash (\ a + a, \quad aT'E', \quad 148) \\
&\vdash (\ + a, \quad T'E', \quad 148) \\
&\vdash (\ + a, \quad E', \quad 1486) \\
&\vdash (\ + a, \quad +TE', \quad 14862) \\
&\vdash (\ a, \quad TE', \quad 14862) \\
&\vdash (\ a, \quad FT'E', \quad 148624) \\
&\vdash (\ a, \quad aT'E', \quad 1486248) \\
&\vdash (\ e, \quad T'E', \quad 1486248) \\
&\vdash (\ e, \quad E', \quad 14862486) \\
&\vdash (\ e, \quad e, \quad 148624863)
\end{aligned}
$$

Nakoniec ešte uvedieme algoritmy na výpočet funkcií FIRST a FOLLOW. V oboch algoritmoch sa na označenie miesta v pravidle, ktoré je z hľadiska ďalšieho postupu zaujímavé, používa bodka. Pritom sa predpokladá, že bodka nie je ani terminálny, ani neterminálny symbol vstupnej gramatiky. Pravidlo gramatiky, do ktorého vložíme bodku na niektoré miesto pravej strany pravidla, nazývame *položka*.

Algoritmus 6.5.

Výpočet funkcie FIRST.

Vstup: bezkontextová gramatika $G = (N, T, P, S)$ a reťazec $\alpha = X_1X_2 \dots X_n \in (N \cup T)^*$.

Výstup: množina $\mathrm{FIRST}(\alpha)$.

Metóda:

1. Vytvoríme množinu F takto:
   a) $F = \{.X_1X_2 \dots X_n\}$,
   b) ak je v množine F prvok, v ktorom je bezprostredne za bodkou neterminálny symbol A, pridáme do množiny F všetky pravidlá z P so sym-

bolom A na ľavej strane a bodku umiestníme pred prvý symbol pravej strany:

$$F = F \cup \{A \to .\delta / B \to \beta.A\gamma \in F,\ A \in N,\ A \to \delta \in P\}$$

Týmto postupom preberieme všetky pravidlá, ktoré sa vyskytnú v deriváciách tvaru

$$A \Rightarrow \delta_1 \Rightarrow \delta_2 \Rightarrow \dots \Rightarrow \delta_n \Rightarrow b\omega$$

kde $\delta_1, \delta_2, \dots, \delta_n$ začínajú neterminálnymi symbolmi, b je terminálny symbol, ktorý patrí do hľadanej množiny FIRST. Pritom uvedená derivácia má tú vlastnosť, že žiadny neterminálny symbol sa neprepisuje pomocou e-pravidiel.

c) ak je v množine F prvok, v ktorom je bodka na konci pravidla, t. j. položka tvaru $B \to \delta.$, vložíme do F nové položky vytvorené tak, že posunieme bodku za B vo všetkých takých položkách v F, kde bodka bola pred B:

$$F = F \cup \{A \to \beta B.\gamma / A \to \beta.B\gamma \in F,\ B \to \delta. \in F\}$$

Položka $B \to \delta.$ sa objaví v F vo dvoch prípadoch:
— $\delta = e$, t. j. ide o e-pravidlo $B \to e$ a v F sa objaví položka $B \to .$ pri vykonávaní kroku 1b,
— $\delta \neq e$, t. j. ide o prípad, keď $\delta = A_1A_2 \dots A_n$, kde $A_1, A_2, \dots, A_n$ sú neterminálne symboly, z ktorých možno generovať prázdne reťazce. Položka $B \to \delta.$ sa v F objaví v dôsledku vykonávania kroku 1a. Vykonaním uvedenej operácie pripravíme preberanie pravidiel v deriváciách tvaru $B\gamma \Rightarrow \gamma \Rightarrow a\omega$, kde začiatočný neterminálny symbol B je prepísaný na prázdny reťazec.

d) kroky b) a c) opakujeme tak dlho, pokiaľ možno do F pridávať ďalšie prvky.

2. Množinu FIRST (α) vytvoríme tak, že do nej vložíme všetky terminálne symboly, ktoré sa vyskytujú bezprostredne za bodkou v niektorom prvku množiny F. Ak je v množine F prvok, v ktorom sa vyskytuje bodka na konci reťazca α, pridáme do FIRST (α) prázdny reťazec:

$$\text{FIRST}(\alpha) = \{a / a \in T,\ A \to \beta.a\gamma \in F\} \cup \{e / \alpha. \in F\}$$

Príklad 6.11. Je daná gramatika $G = (\{E, Z, T, D, F\}, \{+, *, (,), a\}, P, E)$, kde množina P obsahuje pravidlá:

$E \to TZ$ $\qquad$ $D \to *FD \mid e$
$Z \to +TZ \mid e$ $\qquad$ $F \to (E) \mid a$
$T \to FD$

FIRST (E) vypočítame takto:

$$F = \{.E,\ E \to .TZ,\ T \to .FD,\ F \to .(E),\ F \to .a\}$$

FIRST $(E) = \{(, a\}$.
FIRST (Z) vypočítame takto:

$$F = \{.Z,\ Z \to .+TZ,\ Z \to .,\ Z.\}$$

FIRST $(Z) = \{+, e\}$.
FIRST $\{DZ\}$ vypočítame takto:

$$F = \{.DZ,\ D \to .*FD,\ D \to .,\ D.Z,\ Z \to .+TZ,\ Z \to .,\ DZ.\}$$

FIRST $(DZ) = \{*, +, e\}$.

Na podobnom princípe, ako je výpočet funkcie FIRST pomocou algorimu 6.5, môžeme vytvoriť algoritmus na výpočet funkcie FOLLOW.

Algoritmus 6.6.
Výpočet funkcie FOLLOW.
Vstup: bezkontextová gramatika $G = (N, T, P, S)$ a neterminálny symbol A.
Výstup: FOLLOW (A).
Metóda:

1. Výtvoríme množinu $N_e = \{B / B \overset{*}{\Rightarrow} e,\ B \in N\}$, t. j. množinu neterminálnych symbolov, z ktorých možno generovať prázdne reťazce.
2. Vytvoríme množinu F takto:
a) vytvoríme fiktívne pravidlo $A \to A$ a položíme $F = \{A \to A.\}$,
b) ak je v množine F položka, v ktorej je bodka na konci pravidla, t. j. položka tvaru $B \to \gamma.$ a $\gamma \neq e$, vložíme do F nové položky vytvorené tak, že vezmeme všetky pravidlá z P, v ktorých sa na pravých stranách vyskytuje symbol B, a bodku v nich umiestníme práve za tento symbol B:

$$F = F \cup \{C \to \alpha B.\beta / B \to \gamma. \in F,\ \gamma \neq e,\ C \to \alpha B\beta \in P\}$$

Tento krok znamená, že v situácii, keď určujeme FOLLOW (B), je potrebné určiť FIRST (β) pre všetky β v pravidlách $C \to \alpha B\beta$.

c) ak je v množine F prvok, v ktorom je bezprostredne za bodkou neterminálny symbol B, pridáme do množiny F všetky pravidlá z P so symbolom B na ľavej strane a bodku umiestníme pred prvý symbol pravej strany:

$$F = F \cup \{B \to .\alpha / C \to \alpha.B\beta \in F,\ B \in N,\ B \to \alpha \in P\}$$

Tento krok slúži ako súčasť výpočtu FIRST (B), podobne ako krok 1b v predchádzajúcom algoritme.

d) ak je v množine F prvok, v ktorom je bezprostredne za bodkou neter-

minálny symbol patriaci do množiny N_e, pridáme do F ďalšiu položku, ktorú vytvoríme z uvažovanej položky posunutím bodky o jeden symbol doprava:

$$F = F \cup \{A \to \alpha B.\beta / A \to \alpha.B\beta \in F,\ B \in N_e\}$$

Tento krok slúži na výpočet FIRST$(B\beta)$ v prípade, že B môže byť prepísané na prázdny reťazec. Je to obdoba kroku 1c z predchádzajúceho algoritmu.

e) kroky a, c a d opakujeme tak dlho, pokiaľ je možné do F pridávať ďalšie prvky.

3. Množinu FOLLOW(A) vytvoríme tak, že do nej vložíme všetky terminálne symboly, ktoré sa vyskytujú bezprostredne za bodkou v niektorom prvku množiny F. Ak je v množine F prvok, v ktorom sa vyskytuje bodka na konci pravidla a na ľavej strane symbol S (t. j. začiatočný symbol gramatiky), pridáme do FOLLOW(A) prázdny reťazec.

$$\text{FOLLOW}(A) = \{a / a \in T,\ B \to \alpha.a\beta \in F\} \cup \{e / S \to \alpha. \in F\}$$

Príklad 6.12. Je daná gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

$S \to aSAb \mid AB \quad A \to Bb \mid aA \quad B \to bB \mid e$

Množinu FOLLOW(A) určíme pomocou algoritmu 6.6 takto:

1. $N_e = \{B\}$
2. $F = \{$

$A \to A.$	$B \to .$	$A \to .Bb$
$S \to aSA.b$	$B \to .bB$	$A \to .aA$
$S \to A.B$	$S \to AB.$	$A \to B.b$
$A \to aA.$	$S \to aS.Ab$	$\}$

3. FOLLOW$(A) = \{a, b, e\}$.

6.1.1.4 Silné LL(k) gramatiky

Metódu syntaktickej analýzy a postup pri konštrukcii rozkladovej tabuľky pre LL(1) gramatiky môžeme zovšeobecniť na prípad, že syntaktický analyzátor používa na rozhodnutie o tom, akú expanziu urobiť, informáciu o vopred prezeranom reťazci dĺžky k. Gramatiky, pre ktoré možno týmto spôsobom vytvoriť syntaktický analyzátor, sa nazývajú silné LL(k) gramatiky.

Najprv rozšírime definíciu funkcií FIRST a FOLLOW.

Definícia 6.7. Nech je daná bezkontextová gramatika $G = (N, T, P, S)$, reťazec $\alpha \in (N \cup T)^*$ a $A \in N$. Funkcie FIRST$_k$ a FOLLOW$_k$ potom definujeme takto:

$$\text{FIRST}_k(\alpha) = \{x / x \in T^*,\ \alpha \overset{*}{\Rightarrow} xy \text{ a } |x| = k\} \cup \cup \{x / x \in T^*,\ \alpha \overset{*}{\Rightarrow} x \text{ a } |x| < k\}$$

$$\text{FOLLOW}_k(A) = \{x / x \in T^*,\ S \overset{*}{\Rightarrow} wAxy \text{ a } |x| = k\} \cup \cup \{x / x \in T^*,\ S \overset{*}{\Rightarrow} wAx \text{ a } |x| < k\}$$

Ďalej rozšírime definície funkcií FIRST$_k$ a FOLLOW$_k$ tak, aby ich argumentmi mohli byť množiny reťazcov.

Definícia 6.8. Je daná bezkontextová gramatika $G = (N, T, P, S)$ a $R \subset (N \cup T)^*$. Potom

$$\text{FIRST}_k(R) = \{x / x \in \text{FIRST}_k(\alpha) \text{ pre nejaké } \alpha \in R\}$$
$$\text{FOLLOW}_k(R) = \{x / x \in \text{FOLLOW}_k(\alpha) \text{ pre nejaké } \alpha \in R\}$$

Silnú LL(k) gramatiku potom definujeme takto:

Definícia 6.9. Gramatika $G = (N, T, P, S)$ je *silná LL(k) gramatika*, keď pre všetky $A \in N$ platí:
V prípade, že $A \to \alpha$ a $A \to \beta$ sú rôzne pravidlá v P, platí:

$$\text{FIRST}_k(\alpha\text{FOLLOW}_k(A)) \cup \text{FIRST}_k(\beta\text{FOLLOW}_k(A)) = \emptyset$$

Pre silné LL(k) gramatiky môžeme použiť podobné algoritmy na konštrukciu rozkladovej tabuľky aj na syntaktickú analýzu ako pre LL(1) gramatiky (pozri algoritmy 6.2 a 6.4). Predtým, ako uvedieme tieto algoritmy, zavedieme ešte označenie $T^{*k} = \{x / x \in T^*,\ |x| \leq k\}$.

Algoritmus 6.7.

Vytvorenie rozkladovej tabuľky pre silnú LL(k) gramatiku.

Vstup: silná LL(k) gramatika $G = (N, T, P, S)$.

Výstup: rozkladová tabuľka M pre gramatiku G.

Metóda: rozkladová tabuľka M je definovaná na $N \times T^{*k}$.

1. Ak $A \to \alpha$ je i-té pravidlo v P, tak $M(A, x) = \alpha, i$ pre všetky $x \in \text{FIRST}_k(\alpha)$ také, že $|x| = k$.
2. Ak $A \to \alpha$ je i-té pravidlo v P a $y \in \text{FIRST}_k(\alpha)$ také, že $|y| < k$, tak $M(A, z) = \alpha, i$ pre všetky $z \in \text{FIRST}_k(y\text{FOLLOW}_k(A))$.
3. $M(Y, x)$ = chyba vo všetkých ostatných prípadoch.

Príklad 6.13. Vytvoríme rozkladovú tabuľku pre gramatiku $G = (\{S, A\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá:

1. $S \to e$
2. $S \to abA$
3. $A \to Saa$
4. $A \to b$

V tejto gramatike platí:

$$\text{FIRST}_2(e\text{FOLLOW}_2(S)) \cap \text{FIRST}_2(abA\text{FOLLOW}_2(S)) = \text{FOLLOW}_2(S) \cap \text{FIRST}_2(abA) = \{aa, e\} \cap \{ab\} = \emptyset.$$

$\text{FIRST}_2(Saa\text{FOLLOW}_2(A)) \cap \text{FIRST}_2(b\text{FOLLOW}_2(A)) =$
$= \text{FIRST}_2(Saa) \cap \text{FIRST}_2(b\text{FOLLOW}_2(S)) = \{ab, aa\} \cap \{b, ba\} = \emptyset$.

Preto je gramatika G silná LL(2) gramatika a jej rozkladová tabuľka (*tab. 6.7*) má tvar:

Tabuľka 6.7

M	aa	ab	a	ba	bb	b	e
S	e, 1	abA, 2					e, 1
A	Saa, 3	Saa,3		b, 4		b, 4	

Teraz uvedieme algoritmus syntaktickej analýzy pre silné LL(k) gramatiky.

Algoritmus 6.8.
Algoritmus syntaktickej analýzy pre silné LL(k) gramatiky.
Vstup: rozkladová tabuľka M pre silnú LL(k) gramatiku $G = (N, T, P, S)$ a vstupný reťazec $w \in T^*$.
Výstup: ľavý rozklad v prípade, že $w \in L(G)$, inak chybová signalizácia.
Metóda: na začiatku je v zásobníku symbol S, výstupný reťazec je prázdny. Algoritmus vykonáva prechody podľa 1 a 2, kým sa neobjaví situácia podľa 3 alebo 4. Nech $u = \text{FIRST}_k(x)$ je vopred prezeraný reťazec.

1. Expanzia: $(x, A\alpha, \pi) \vdash (x, \beta\alpha, \pi i)$, ak $M(A, u) = \beta, i$. Symbol A na vrchu zásobníka je nahradený reťazcom β a číslo pravidla i sa pripojí k výstupnému reťazcu.
2. Porovnanie: $(ax, a\alpha, \pi) \vdash (x, \alpha, \pi)$, ak $a \in T$. Ak symbol a na vrchu zásobníka je rovnaký ako prvý symbol vopred prezeraného reťazca, odstráni sa zo zásobníka a vstupný symbol sa prečíta.
3. Prijatie: Ak sme v konfigurácii (e, e, π), analýza končí a výstupný reťazec π je ľavý rozklad vstupného reťazca.
4. Chyba: Vo všetkých ostatných prípadoch analýza končí chybovou signalizáciou.

Príklad 6.14. Urobíme analýzu reťazca *ababbaa* použitím rozkladovej tabuľky pre gramatiku z príkladu 6.13.

$$
\begin{aligned}
(ababbaa, S, e) &\vdash (ababbaa, abA, 2)\\
&\vdash (babbaa, bA, 2)\\
&\vdash (abbaa, A, 2)\\
&\vdash (abbaa, Saa, 23)\\
&\vdash (abbaa, abAaa, 232)\\
&\vdash (bbaa, bAaa, 232)\\
&\vdash (baa, Aaa, 232)
\end{aligned}
$$

$$
\begin{aligned}
&\vdash (baa, baa, 2324)\\
&\vdash (aa, aa, 2324)\\
&\vdash (a, a, 2324)\\
&\vdash (e, e, 2324)
\end{aligned}
$$

6.1.2 Slabé LL gramatiky

Pre všetky doteraz preberané špeciálne prípady LL(k) gramatík sme pri syntaktickej analýze využívali iba informácie o symbole na vrchu zásobníka a o k symboloch z doteraz neanalyzovanej časti vstupného reťazca. V prípade slabých LL(k) gramatík už s týmito informáciami nevystačíme a musíme pri rozhodovaní o ďalšom postupe analýzy využiť informáciu o jej priebehu. Takúto situáciu si ukážeme na príklade.

Príklad 6.15. Je daná gramatika $G = (\{S, A\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

$S \rightarrow aAaa \mid bAba$
$A \rightarrow b \mid e$

Ak máme vykonať expanziu neterminálneho symbolu A za predpokladu, že vopred prezeraný reťazec je ba, nie je možné rozhodnúť, či použiť pravidlo $A \rightarrow b$ alebo $A \rightarrow e$. Rozhodnutie o výbere jedného z týchto pravidiel možno urobiť zo znalosti predchádzajúceho postupu analýzy. Ak bol v predchádzajúcom kroku prečítaný symbol a, tak použijeme pravidlo $A \rightarrow b$, ak bol prečítaný symbol b, použijeme pravidlo $A \rightarrow e$.

Uvedená gramatika nie je silná LL(2) gramatika, pretože platí $\text{FIRST}_2(b\text{FOLLOW}_2(A)) \cap \text{FIRST}_2(\text{FOLLOW}_2(A)) = \{ba\}$. Pritom z predchádzajúcej úvahy vyplýva, že gramatika je LL(2) gramatika, čo vyplýva aj z nasledujúcej definície LL(k) gramatík.

Definícia 6.10. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Hovoríme, že G je *LL(k) gramatika* pre nejaké pevné celé číslo k, ($k \geqq 0$), ak v prípade existencie dvoch ľavých derivácii

$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\beta\alpha \overset{*}{\Rightarrow} wx$
$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\gamma\alpha \overset{*}{\Rightarrow} wy$

takých, že $\text{FIRST}_k(x) = \text{FIRST}_k(y)$, platí $\beta = \gamma$.

Neformálne povedané, gramatika G je LL(k), ak pri danom reťazci $wA\alpha \in (N \cup T)^*$ a daných prvých k terminálnych symboloch (ak existujú), derivovateľných z $A\alpha$, existuje najviac jedno pravidlo, ktoré možno použiť na expanziu symbolu A, a tak získať deriváciu ľubovoľného terminálneho reťazca

začínajúceho reťazcom w, za ktorým nasleduje uvedených k symbolov, ktoré možno derivovať z $A\alpha$.

Z tejto definície LL(k) gramatiky vyplýva, že pri jednoznačnom určení pravidla na expanziu neterminálneho symbolu A sa využíva informácia o k nasledujúcich symboloch vo vstupnom reťazci ($\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(y)$) a o doterajšom priebehu analýzy. Všimnite si, že už analyzovaná časť vstupného reťazca w je v oboch deriváciach rovnaká a jej analýza musela prebehnúť rovnakým spôsobom.

História syntaktickej analýzy je reprezentovaná obsahom zásobníka. Reťazec, ktorý je v zásobníku, budeme nazývať perspektívna prípona.

Definícia 6.11. Predpokladajme, že $S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\beta\alpha$ je ľavá derivácia v gramatike $G = (N, T, P, S)$. Reťazec γ nazývame *perspektívna prípona* v G, ak $\gamma = S$, alebo γ je príponou reťazca $\beta\alpha$.

To znamená, že γ je reťazec, ktorý je takou príponou ľavej vetnej formy, že nesiaha pred žiadny symbol vložený do ľavej vetnej formy pri poslednej vykonanej expanzii.

Ak perspektívna prípona začína neterminálnym symbolom, budeme ju nazývať *úplná perspektívna prípona*. V okamihu, keď je v zásobníku úplná perspektívna prípona, bude syntaktický analyzátor vykonávať operáciu expanzie.

Príklad 6.16. Ukážeme, aké reťazce budú obsahom zásobníka pri analýze dvoch reťazcov generovaných gramatikou z príkladu 6.15 (*tab. 6.8* a *6.9*).

Tabuľka 6.8

vstupný reťazec	obsah zásobníka	vykonaná operácia
bba	S	expanzia $S \rightarrow bAba$
bba	$bAba$	porovnanie b
ba	Aba	expanzia $A \rightarrow e$
ba	ba	porovnanie b
a	a	porovnanie a
e	e	prijatie

Tabuľka 6.9

vstupný reťazec	obsah zásobníka	vykonaná operácia
$abaa$	S	expanzia $S \rightarrow aAaa$
$abaa$	$aAaa$	porovnanie a
baa	Aaa	expanzia $A \rightarrow b$
baa	baa	porovnanie b
aa	aa	porovnanie a
a	a	porovnanie a
e	e	prijatie

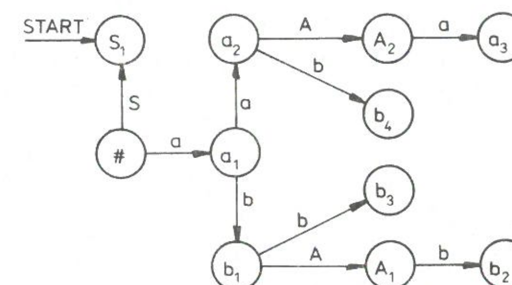
Z uvedených dvoch príkladov je vidieť, že výber pravidla na expanziu symbolu A v prípade, že vopred prezeraný reťazec je ba, možno vykonať na základe znalosti obsahu zásobníka a vopred prezeraného reťazca.

Nasledujúca tabuľka ukazuje (*tab. 6.10*), aké pravidlo sa vyberie pri expanzii na základe informácie o vopred prezeranom reťazci dĺžky 2 a obsahu zásobníka.

Tabuľka 6.10

	aa	ab	ba	bb
S	$S \rightarrow aAaa$	$S \rightarrow aAaa$		$S \rightarrow bAba$
Aaa	$A \rightarrow e$		$A \rightarrow b$	
Aba			$A \rightarrow e$	$A \rightarrow b$

Tento príklad je jednoduchý, pretože počet všetkých reťazcov, ktoré môžu byť v zásobníku pri expanzii (úplných perspektívnych prípon), je konečný. Úplných perspektívnych prípon však môže byť nekonečne veľa. Dá sa dokázať, že množina úplných perspektívnych prípon tvorí pre danú gramatiku regulárny jazyk. To znamená, že je možné pre analýzu perspektívnych prípon vytvoriť konečný automat. Takýto automat sa nazýva *charakteristický konečný automat* LL analyzátora. Skrátene ho budeme nazývať *LL automat*. Prechodový diagram LL automatu pre gramatiku z príkladu 6.15 je na *obr. 6.1*.



Obr. 6.1. Charakteristický LL automat pre gramatiku s pravidlami $S' \rightarrow S$, $S \rightarrow aAaa \mid bAba$, $A \rightarrow b \mid e$

Priebeh syntaktickej analýzy môžeme znázorniť pomocou činnosti LL automatu. Pokiaľ sa bude pri analýze do zásobníka niečo pridávať, bude automat vykonávať prechody v smere šípok, pokiaľ sa bude niečo zo zásobníka vyberať, prejaví sa to pohybom proti smeru šípok.

Ukážme činnosť automatu na *obr. 6.1* pri analýze reťazca $abaa$. Na začiatku je automat v stave S_1, čo znamená, že v zásobníku je reťazec S. Tento reťazec tvorí úplnú perspektívnu príponu, a preto sa bude vykonávať expanzia. Pravidlo vybraté na expanziu je $S \rightarrow aAaa$.

Najprv sa zo zásobníka vylúči S, čo sa prejaví tým, že sa prejde do stavu $\#$. Zo stavu $\#$ sa prejde po ceste, ktorá zodpovedá obrátenej pravej strane pravidla $S \rightarrow aAaa$. Prejde sa do stavu a_3. V tomto stave je na vrchu zásobníka symbol a. Ďalej sa vykoná operácia porovnania a prejde sa do stavu A_2. V tomto okamihu je v zásobníku úplna perspektívna prípona a vykoná sa expanzia. Pretože stav A_2 zodpovedá prípone Aaa a vopred prezeraný reťazec je ba, vyberie sa na expanziu pravidlo $A \rightarrow b$. Zo zásobníka sa vylúči symbol A a prejde sa do stavu a_2. Ďalej sa do zásobníka vloží symbol b a prejde sa do stavu b_4. Ďalší postup analýzy spočíva vo vykonaní troch operácií porovnania pre symboly b, a, a a ukončenia analýzy v stave $\#$.

Z doterajších úvah o syntaktickej analýze slabých LL(k) gramatík vyplýva, že základom syntaktického analyzátora je LL automat. Na vytvorenie tohto automatu je možné použiť tri metódy:

1. Vytvorenie súboru množín LL(k) položiek. Funkcia GOTO definovaná na tejto množine je prechodová funkcia LL automatu.
2. Vytvorenie LL automatu priamo na základe gramatiky.
3. Vytvorenie sústavy regulárnych rovníc, ktorých riešením získame regulárne výrazy opisujúce množiny úplných perspektívnych prípon, a potom vytvorenie konečného automatu pre tieto regulárne výrazy.

Ďalej sa budeme zaoberať len prvou metódou konštrukcie LL automatu, založenou na vytvorení súboru množín LL(k) položiek.

Definícia 6.12. LL(k) položka v gramatike $G = (N, T, P, S)$ je objekt tvaru $[A \rightarrow \alpha . \beta, u]$, kde $A \rightarrow \alpha\beta$ je pravidlo v P, u je reťazec vo $\mathrm{FIRST}_k(\beta/\mathrm{FOLLOW}_k(A))$.

Súbor množín LL(k) položiek pre danú gramatiku vytvoríme pomocou nasledujúceho algoritmu.

Algoritmus 6.9.

Vytvorenie súboru množín LL(k) položiek.

Vstup: bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: súbor $\mathscr{L}$ množín LL(k) položiek pre G.

Metóda:

1. Ku gramatike G vytvoríme rozšírenú gramatiku $G' = (N \cup \{S'\}, T, P \cup \{S' \rightarrow S\}, S')$, kde S' je nový neterminálny symbol.
2. Začiatočnú množinu LL(k) položiek D_0 vytvoríme takto:
   a) $D_0 = \{[S' \rightarrow S., e]\}$,
   b) ak $[A \rightarrow \alpha B., u] \in D_0$, $B \in N$ a $B \rightarrow \gamma \in P$, tak $D_0 = D_0 \cup \{[B \rightarrow \gamma., u]\}$,
   c) krok 2b opakujeme tak dlho, kým do množiny D_0 možno pridávať ďalšie položky.
   d) $\mathscr{L} = \{D_0\}$.
3. Ak sme skonštruovali množinu LL(k) položiek D_i, tak skonštruujeme pre každý symbol $X \in N \cup T$ taký, že leží v niektorej položke v D_i pred bodkou, ďalšiu množinu D_j, kde j je index väčší ako najväčší index doteraz vytvorenej množiny D_i, takto:
   a) $D_j = \{[A \rightarrow \alpha.X\beta, v]/[A \rightarrow \alpha X.\beta, u] \in D_i, v \in \mathrm{FIRST}_k(Xu)\}$,
   b) ak $[A \rightarrow \alpha B.\beta, u] \in D_j$, $B \in N$, $B \rightarrow \gamma \in P$, tak $D_j = D_j \cup \{[B \rightarrow \gamma., u]$,
   c) krok 3b opakujeme dovtedy, kým možno do D_j pridávať ďalšie položky,
   d) $\mathscr{L} = \mathscr{L} \cup \{D_j\}$.
4. Krok 3 opakujeme pre všetky vytvorené množiny dovtedy, kým možno do $\mathscr{L}$ pridávať ďalšie nové množiny.

Príklad 7.17. Pomocou algoritmu 6.9 vytvoríme súbor množín LL(2) položiek pre gramatiku z príkladu 6.15.

Po rozšírení bude mať gramatika pravidlá:

$$S' \rightarrow S$$
$$S \rightarrow aAaa \mid bAba$$
$$A \rightarrow b \mid e$$

Súbor množín LL(2) položiek bude mať tvar:

$$\begin{array}{ll}
D_0 = \{[S' \rightarrow S., e] & D_4 = \{[S \rightarrow bA.ba, ba] \\
\quad [S \rightarrow aAaa., e] & \quad [A \rightarrow b., ba] \\
\quad [S \rightarrow bAba., e]\} & \quad [A \rightarrow ., ba]\} \\
D_1 = \{[S' \rightarrow .S, aa] & D_5 = \{[S \rightarrow a.Aaa, ba] \\
\quad [S' \rightarrow .S, ab] & \quad [S \rightarrow a.Aaa, aa]\} \\
\quad [S' \rightarrow .S, bb] & D_6 = \{[A \rightarrow .b, ba]\} \\
D_2 = \{[S \rightarrow aAa.a, a] & D_7 = \{[S \rightarrow b.Aba, ba] \\
\quad [S \rightarrow bAb.a, a]\} & \quad [S \rightarrow b.Aba, bb]\} \\
D_3 = \{[S \rightarrow aA.aa, aa] & D_8 = \{[A \rightarrow .b, bb]\} \\
\quad [A \rightarrow b., aa] & D_9 = \{[S \rightarrow .aAaa, ab] \\
\quad [A \rightarrow ., aa]\} & \quad [S \rightarrow .aAaa, aa]\} \\
 & D_{10} = \{[S \rightarrow .bAba, bb]\}
\end{array}$$

Na súbore množín LL(k) položiek budeme definovať funkciu GOTO.

Definícia 6.13. Ak je daný súbor množín LL(k) položiek pre gramatiku $G = (N, T, P, S)$, tak *funkcia* GOTO je definovaná takto:

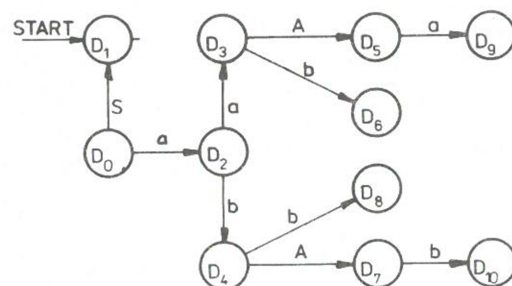
$$\mathrm{GOTO}(D_i, X) = D_j, \text{ ak } [A \rightarrow \alpha X.\beta, v] \in D_i, [A \rightarrow \alpha.X\beta, u] \in D_j \text{ a } u \in \mathrm{FIRST}_k(Xv)$$

Funkciu GOTO môžeme rozšíriť tak, aby druhým argumentom mohol byť i reťazec, takto:

$$\mathrm{GOTO}(D_i, e) = D_i$$
$$\mathrm{GOTO}(D_i, \alpha X) = \mathrm{GOTO}(\mathrm{GOTO}(D_i, \alpha), X)$$

Pre súbor $\mathscr{L}$ množín LL(k) položiek môžeme znázorniť funkciu GOTO vo forme prechodového diagramu. Vrcholy tohto diagramu zodpovedajú množinám zo súboru $\mathscr{L}$ a ak GOTO $(D_i, X) = D_j$, tak z vrchola D_i do vrchola D_j vedie hrana s ohodnotením X. Tento graf nazývame GOTO graf.

GOTO graf pre súbor množín LL(k) položiek z príkladu 6.17 je na *obr. 6.2*.



Obr. 6.2. GOTO graf pre gramatiku s pravidlami $S' \to S$, $S \to aAaa \mid bAba$, $A \to e \mid b$

Pri porovnávaní tohto grafu s grafom na *obr. 6.1* zistíme, že GOTO graf je okrem ohodnotenia vrcholov totožný s prechodovým diagramom charakteristického LL automatu pre danú gramatiku.

Súbor množín LL(k) položiek slúži nielen ako východisko pri konštrukcii LL automatu, ale môžeme ho použiť pri konštrukcii rozkladovej tabuľky. Rozkladovú tabuľku môžeme zostrojiť na základe súboru množín LL(k) položiek pomocou nasledujúceho algoritmu.

Algoritmus 6.10.
Rozkladová tabuľka pre LL(k) gramatiku.
Vstup: LL(k) gramatika $G = (N, T, P, S)$ a súbor $\mathscr{L}$ množínLL(k) položiek pre G.
Výstup: rozkladová tabuľka M pre gramatiku G.
Metóda: rozkladová tabuľka je definovaná na $Q \times T^{*k}$, kde

$$Q = \{D_i / A \to \alpha.B\beta \in D_i,\ B \in N\}$$

1. Ak $[A \to \alpha B.\beta, u] \in D_i$, tak $M(D_j, v) = \gamma, n$ ak platí:
   a) GOTO $(D_i, B) = D_j$,
   b) GOTO $(D_i, \gamma^R) = D_r$ a $[B \to \gamma., u] \in D_i$, $[B \to .\gamma, v] \in D_r$,
   c) $B \to \gamma$ je n-té pravidlo z P.
2. V ostatných prípadoch $M(D_j, v) =$ chyba.

Príklad 6.18. Vytvoríme rozkladovú tabuľku na základe súboru množín LL(k) položiek z príkladu 6.17. Jednotlivé pravidlá z P očíslujeme takto:

1. $S \to aAaa$
2. $S \to bAba$
3. $A \to b$
4. $A \to e$

Rozkladová tabuľka (*tab. 6.11*) bude mať tvar:

tabuľka 6.11

M	aa	ab	a	ba	bb	b	e
D_1	$aAaa$, 1	$aAaa$, 1			$bAba$, 2		
D_5	e, 4			b, 3			
D_7				e, 4	b, 3		

Syntaktickú analýzu pre slabé LL(k) gramatiky môžeme robiť dvoma spôsobmi:

1. Pomocou rovnakého algoritmu ako pre silné LL(k) gramatiky (pozri algoritmus 6.8).
2. Pomocou algoritmu, ktorý využíva LL automat a rozkladovú tabuľku.

V prípade, že budeme chcieť použiť algoritmus pre silné LL(k) gramatiky, musíme rozkladovú tabuľku upraviť tak, aby sa do zásobníka pri expanziách namiesto neterminálnych symbolov ukladalo ohodnotenie príslušných stavov LL automatu. Prakticky to znamená, že v prípade, keď $M(D_i, u) = \alpha A\beta, i$, kde $D_i =$ GOTO (D_j, B) a $B \to \alpha A\beta \in P$, musíme reťazec $\alpha A\beta$, nahradiť reťazcom $\alpha D_r\beta$, kde $D_r =$ GOTO $(D_j, \beta^R A)$. Začiatočný symbol, ktorý je na začiatku analýzy umiestený v zásobníku, je symbol D_i, ak $[S' \to .S, u] \in D_i$.

Príklad 6.19. Po uvedenej úprave bude mať rozkladová tabuľka z príkladu 6.18 (*tab. 6.12*) tvar:

Tabuľka 6.12

M	aa	ab	a	ba	bb	b	e
D_1	aD_5aa, 1	aD_5aa, 1			bD_7ba, 2		
D_5	e,			b, 3			
D_7				e, 4	b, 3		

Pomocou algoritmu 6.8 urobíme teraz analýzu reťazca *abaa*:

$$
\begin{aligned}
(abaa, D_1, e) &\vdash (\ abaa,\ aD_5aa,\ 1)\\
&\vdash (\ baa,\ D_5aa,\ 1)\\
&\vdash (\ baa,\ baa,\ 13)\\
&\vdash (\ aa,\ aa,\ 13)\\
&\vdash (\ a,\ a,\ 13)\\
&\vdash (\ e,\ e,\ 13)
\end{aligned}
$$

6.1.3 Vlastnosti LL gramatík a jazykov

V predchádzajúcich článkoch sme sa zaoberali metódami syntaktickej analýzy pre jednotlivé triedy LL(k) gramatík. V tomto odseku uvedieme niektoré vlastnosti LL(k) gramatík a zmienime sa o teoretických problémoch, ktoré ovplyvňujú praktickú použiteľnosť teórie LL(k) gramatík pri konštrukcii syntaktických analyzátorov bezkontextových jazykov. Najprv uvedieme definíciu LL jazykov.

Definícia 6.14. Bezkontextový jazyk L sa nazýva *LL(k) jazyk*, ak existuje LL(k) gramatika G taká, že $L = L(G)$.

Bezkontextový jazyk L sa nazýva *LL jazyk*, ak existuje LL(k) gramatika G pre nejaké $k \geqq 0$ taká, že $L = L(G)$.

Teraz uvedieme niekoľko vlastností LL(k) gramatík.

Veta 6.1. Každá LL(k) gramatika je jednoznačná.

Jednoznačnosť LL(k) gramatiky vyplýva z definície (pozri definíciu 6.10). Ak je daná ľavá derivácia

$$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\beta\alpha \overset{*}{\Rightarrow} wx$$

môže byť v gramatike najviac jedno pravidlo $A \to \beta$ také, že $\text{FIRST}_k(\beta\alpha)$ obsahuje $\text{FIRST}_k(x)$. Pritom je zrejmé, že reťazec α je jednoznačne určený predponou ľavej vetnej formy wA. To znamená, že pravidlo $A \to \beta$ na expanziu symbolu A vo vetnej forme $wA\alpha$ je jednoznačne určené reťazcom w a množinou $\text{FIRST}_k(x)$. Pre každú vetu generovanú LL(k) gramatikou potom existuje jediná ľavá derivácia, a teda aj jediný derivačný strom.

Veta 6.2. Pre dve gramatiky $G_1 = (N_1, T_1, P_1, S_1)$ a $G_2 = (N_2, T_2, P_2, S_2)$ je rozhodnuteľné, či sú ekvivalentné, t. j. či $L(G_1) = L(G_2)$.

Veta 6.3. Žiadna LL(k) gramatika nie je zľava rekurzívna.

Predpokladajme, že gramatika $G = (N, T, P, S)$ obsahuje zľava rekurzívny neterminálny symbol A. Potom existuje derivácia $A \overset{+}{\Rightarrow} A\alpha$. Ak $\alpha \overset{*}{\Rightarrow} e$, tak gramatika obsahuje cyklus a nie je jednoznačná, a teda nemôže byť LL(k) pre žiadne k.

Predpokladajme, že $\alpha \overset{*}{\Rightarrow} v$ pre nejaké $v \in T^+$. Ďalej môžeme predpokladať, že $A \overset{*}{\Rightarrow} u$ pre nejaké $u \in T^*$ a že existuje ľavá derivácia

$$S \overset{*}{\Rightarrow} wA\delta \overset{*}{\Rightarrow} wA\alpha^k\delta \overset{*}{\Rightarrow} wuv^kx$$

Ďalej existuje iná ľavá derivácia

$$S \overset{*}{\Rightarrow} wA\delta \overset{*}{\Rightarrow} wA\alpha^k\delta \overset{*}{\Rightarrow} wA\alpha^{k+1}\delta \overset{*}{\Rightarrow} wuv^{k+1}x$$

Pretože $\text{FIRST}_k(uv^kx) = \text{FIRST}_k(uv^{k+1}x)$, dostávame sa do sporu s definíciou LL(k) gramatiky pre uvedené dve derivácie a pre ľubovoľné k.

Veta 6.4. Pre danú bezkontextovú gramatiku G a dané pevné $k \geqq 0$ je rozhodnuteľné, či G je alebo nie je LL(k).

Pre LL(1) gramatiky stačí preveriť pravidlá gramatiky podľa definície 6.3, pre silné LL(k) gramatiky podľa definície 6.9.

Na rozhodnutie o tom, či bezkontextová gramatika je slabá LL(k) gramatika, nemožno použiť definíciu 6.10, pretože počet derivácií, ktoré by bolo potrebné preveriť, môže byť nekonečný. V tomto prípade možno použiť tento postup (pre danú gramatiku a pre dané k):

a) vytvoriť súbor množín LL(k) položiek pomocou algoritmu 6.9,

b) vytvoriť rozkladovú tabuľku pomocou algoritmu 6.10,

c) preveriť, či je v rozkladovej tabuľke pre každú položku definovaná jediná expanzia alebo chyba.

V prípade, že pri konštrukcii rozkladovej tabuľky dôjde k tomu, že v jednej položke $M(X, w)$ sa objavia dve rôzne pravidlá, daná gramatika nie je LL(k) pre dané k.

Veta 6.5. Pre danú bezkontextovú gramatiku G je nerozhodnuteľné, či je LL(k) gramatikou pre nejaké $k \geqq 0$.

Poznámka 6.3. Môžeme si predstaviť taký postup, že položíme $k = 0$ a budeme zisťovať, či gramatika je LL(k). Ak bude odpoveď záporná, zväčšíme k o jednotku a postup opakujeme. Tento proces však môže prebiehať donekonečna, pretože k sa môže ľubovoľne zväčšovať.

Veta 6.6. Ak daná bezkontextová gramatika G, ktorá nie je LL(k) pre dané pevné k, je nerozhodnuteľné, či ku G existuje ekvivalentná gramatika, ktorá je LL(k).

Posledné dve vety predstavujú základné teoretické problémy LL gramatík, z ktorých vyplýva celý rad ťažkostí pri konštrukcii syntaktických analyzátorov pracujúcich metódou zhora nadol. Nemožno vytvoriť algoritmy, ktoré by zistili, či daná gramatika je LL(k) pre nejaké k a nemožno tiež vytvoriť algoritmus, ktorý by našiel k danej gramtike ekvivalentnú LL(k) gramatiku, dokonca ani v prípade, že taká gramatika existuje. Odhliadnúc od týchto problémov, existuje celý rad transformácií bezkontextových gramatík, ktoré vedú k LL gramatikám. Niektoré z týchto transformácií si v ďalšom uvedieme.

Zaujímavou skutočnosťou teórie LL gramatík je, že každá LL(1) gramatika je silná LL(1) gramatika. To znamená, že definícia silnej LL(k) gramatiky a definícia všeobecnej (slabej) LL(k) gramatiky sú pre $k = 1$ ekvivalentné.

Veta 6.7. Bezkontextová gramatika $G = (N, T, P, S)$ je LL(1) gramatikou práve vtedy, ak je silnou LL(1) gramatikou, t. j. keď pre každý neterminálny symbol A platí: ak $A \to \beta$ a $A \to \gamma$ sú rôzne pravidlá v P, tak platí $\text{FIRST}_1(\beta\text{FOLLOW}_1(A)) \cap \text{FIRST}_1(\gamma\text{FOLLOW}_1(A)) = \emptyset$.

Predpokladajme, že A, β, γ existujú tak, ako je uvedené vo vete, ale

$\mathsf{FIRST}_1(\beta/\mathsf{FOLLOW}_1(A)) \cap \mathsf{FIRST}_1(\gamma/\mathsf{FOLLOW}_1(A))$ obsahuje $\{x\}$

Potom podľa definície funkcií FIRST aFOLLOW existujú derivácie

$$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\beta\alpha \overset{*}{\Rightarrow} wxy$$
$$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\gamma\alpha \overset{*}{\Rightarrow} wxz$$

pre nejaké y a z. Pretože $\gamma \neq \beta$, gramatika G nie je LL(1).

Predpokladajme, že G nie je LL(1). Potom existujú dve derivácie:

$$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\beta\alpha \overset{*}{\Rightarrow} wx$$
$$S \overset{*}{\Rightarrow} wA\alpha \Rightarrow w\gamma\alpha \overset{*}{\Rightarrow} wy$$

také, že platí $\mathsf{FIRST}_1(x) \cap \mathsf{FIRST}_1(y) \neq \emptyset$, ale $\beta \neq \gamma$. Potom $A \to \beta$ a $A \to \gamma$ sú rôzne pravidlá v P a

$\mathsf{FIRST}_1(\beta/\mathsf{FOLLOW}_1(A)) \cap \mathsf{FIRST}_1(\gamma/\mathsf{FOLLOW}_1(A))$

obsahuje reťazec z $\mathsf{FIRST}_1(x)$, ktorý je aj vo $\mathsf{FIRST}_1(y)$.

Ako dôsledok vety 6.7 možno dokázať, že gramatika G je LL(1) vtedy, keď pre každú množinu pravidiel tvaru $A \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$ platia tieto podmienky:

1. $\mathsf{FIRST}_1(\alpha_1)$, $\mathsf{FIRST}_1(\alpha_2)$, ..., $\mathsf{FIRST}_1(\alpha_n)$ sú po dvoch disjunktné množiny.
2. Ak $\alpha_i \overset{*}{\Rightarrow} e$, tak $\mathsf{FIRST}_1(\alpha_j) \cap \mathsf{FOLLOW}_1(A) = \emptyset$ pre $1 \leqq j \leqq n$, $i \neq j$.

Ďalej sa budeme zaoberať vlastnosťami LL(k) jazykov. Najskôr uvedieme niekoľko príkladov jazykových konštrukcií, ktoré nemožno opísať pomocou LL(k) gramatík.

Príklad 6.20. Najjednoduchším príkladom jazyka, ktorý nie je LL(k) pre žiadne $k \geqq 0$, je jazyk $L_1 = \{a^n b^n / n \geqq 0\} \cup \{a^n c^n / n \geqq 0\}$.

Jazyk L_1 predstavuje symetrickú zátvorkovú štruktúru, v ktorej k ľavej zátvorke prislúcha alternatívna pravá zátvorka, pričom pravé zátvorky musia byť v určitom reťazci vždy rovnaké. Informáciu o tom, aké pravé zátvorky použiť, možno zistiť až po prečítaní n ľavých zátvoriek. Pretože n je ľubovoľné číslo, možno nájsť také n, pre ktoré je $n > k$, kde k je pevne dané číslo.

Dá sa ukázať, že gramatika, ktorá generuje jazyk L_1, môže byť napr.

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

kde P obsahuje pravidlá

$$S \to A | B$$
$$A \to aAb | e$$
$$B \to aBc | e$$

Táto gramatika nie je LL(k), pretože platí $\mathsf{FIRST}_k(A) \cap \mathsf{FIRST}_k(B) = \{a^k\}$ pre ľubovoľné $k \geqq 1$. Pritom sa nedá pre jazyk L_1 vytvoriť žiadna LL(k) gramatika.

Jazyk $L_2 = \{a^n w / n \geqq 0, w \in \{b, c\}^n\}$ sa veľmi podobá jazyku L_1. Rozdiel je iba

v tom, že v jazyku L_2 sa môžu vyskytovať zátvorky ľubovoľného typu v každom reťazci. Pritom platí $L_1 \subset L_2$.

Jazyk L_2 je LL(1) jazyk a LL(1) gramatika, ktorá ho generuje je $G = (\{S, A\}, \{a, b, c\}, \{S \to aSa | e, A \to b | c\}, S)$.

Príklad 6.21. Iný príklad jazyka, ktorý nie je LL(k) pre žiadne k uvádza Backhouse [2]. Ide o aritmetické a boolovské výrazy generované gramatikou s týmito pravidlami:

⟨výraz⟩ → ⟨boolovský výraz⟩ | ⟨aritmetický výraz⟩
⟨boolovský výraz⟩ → ⟨boolovský člen⟩
⟨zvyšok boolovského výrazu⟩
⟨zvyšok boolovského výrazu⟩ → e | or⟨boolovský člen⟩
⟨boolovský člen⟩ → TRUE | FALSE | (⟨boolovský výraz⟩)
⟨aritmetický výraz⟩ → ⟨aritmetický člen⟩
⟨zvyšok aritmetického výrazu⟩
⟨zvyšok aritmetického výrazu⟩ → e | + ⟨aritmetický člen⟩
⟨aritmetický člen⟩ → 0 | 1 | (⟨aritmetický výraz⟩)

Táto gramatika je jednoznačná, nie je zľava rekurzívna a pritom nie je LL(k) pre žiadne k a ani sa na LL(k) gramatiku nedá transformovať. Dôvodom je skutočnosť, že v tejto gramatike sú možné napríklad nasledujúce dve derivácie:

⟨výraz⟩ ⇒ ⟨boolovský výraz⟩ $\overset{*}{\Rightarrow}$ (((... (TRUE) ...)))
⟨výraz⟩ ⇒ ⟨aritmetický výraz $\overset{*}{\Rightarrow}$ (((... (0) ...)))

Z týchto derivácii vidieť, že druh výrazu (t. j. aritmetický alebo boolovský) sa dá zistiť až po prečítaní úvodnej série ľavých zátvoriek, ktorá môže byť ľubovoľne dlhá. V gramatike platí:

FIRST_k(⟨boolovský výraz}) ∩ FIRST_k(⟨aritmetický výraz⟩) = $\{(^k\}$ pre ľubovoľné $k \geqq 1$.

Príklad 6.22. Ďalší typ jazykovej konštrukcie, ktorá nie je LL(k), opisuje gramatika $G = (\{S, A, B\}, \{0, 1, a, b\}, P, S)$, kde P obsahuje pravidla

$$S \to A | B$$
$$A \to aAb | 0$$
$$B \to aBbb | 1$$

$$L(G) = \{a^n 0 b^n / n \geqq 0\} \cup \{a^n 1 b^{2n} / n \geqq 0\}.$$

Ďalej ukážeme, že LL(k) jazyky tvoria vlastnú podmnožinu LL($k + 1$) jazykov. To znamená, že existujú jazyky, ktoré sú LL($k + 1$), ale pritom nie sú LL(k).

Príklad 6.23. Ukážeme jazyk, ktorý možno generovať pomocou LL(k) gramatiky. $L_k = \{a^n w / n \geqq 1, w \in \{b, c, b^k d\}^n\}$. Jazyk L_k je LL(k) jazyk, ale pritom nie je LL($k - 1$) jazyk pre $k \geqq 1$. Jazyk L_k možno generovať gramatikou

$G_1 = (\{S, T, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidlá

$S \to aT$ $\qquad T \to SA \mid A$
$A \to bB \mid c$ $\qquad B \to b^{k-1}\, d \mid e$

Táto gramatika je LL(k) gramatika, pretože okrem iného platí $\mathrm{FIRST}_k(b^{k-1}d) \cap \mathrm{FOLLOW}_k(B) = \emptyset$ pre $k \geqq 1$, nie je však LL($k-1$) gramatikou, pretože $\mathrm{FIRST}_{k-1}(b^{k-1}d) \cap \mathrm{FOLLOW}_{k-1}(B) = \{b^{k-1}\}$.

Dá sa ukázať, že ku gramatike G_1 neexistuje ekvivalentná LL($k-1$) gramatika. Jazyk L_k sa dá generovať aj gramatikou

$G_2 = (\{S, A\}, \{a, b, c, d\}, P, S)$

kde P obsahuje pravidlá

$S \to aSA \mid aA$ $\qquad A \to b^k d \mid b \mid c$

Táto gramatika je ekvivalentná s gramatikou G_1 a je LL($k+1$) gramatikou. Z príkladu 6.23 vidno, že množina LL(k) jazykov je vlastnou podmnožinou LL($k+1$) jazykov pre ľubovoľné $k \geqq 1$.

6.1.4 Transformácie LL gramatík

V tomto článku uvedieme dve transformácie LL gramatík, ktoré vedú k jednoduchším gramatikám. Prvá transformácia je transformácia LL(1) gramatiky na q-gramatiku, druhá transformácia je transformácia slabej LL(k) gramatiky na silnú LL(k) gramatiku.

Najprv ukážeme, že operácia „substitúcie" zachováva vlastnosť gramatiky byť LL(k) gramatikou.

Veta 6.8. Nech $G = (N, T, P, S)$ je bezkontextová gramatika s pravidlom $A \to B\alpha$ v P, kde $B \in N$, a $B \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ sú všetky pravidlá v P so symbolom B na ľavej strane. Gramatiku $G_1 = (N, T, P_1, S)$ vytvoríme z G vylúčením pravidla $A \to B\alpha$ z P a pridaním pravidiel $A \to \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_m\alpha$. Potom $L(G) = L(G_1)$ a ak G je LL(k) gramatika, tak aj G_1 je LL(k) gramatika. Túto transformáciu budeme nazývať *rohová substitúcia*.

Jazyk $L(G_1) = L(G)$, pretože ide o transformáciu v zmysle vety 4.3. Stačí teda dokázať, že G_1 je LL(k) gramatika, ak G je LL(k) gramatika. Všimnime si, že derivácie v G_1 sú v zásade rovnaké ako v G s výnimkou prípadu, keď sú v derivácii použité pravidlá $A \to B\alpha$ a $B \to \beta_i$. Vtedy je v gramatike G_1 použité pravidlo $A \to \beta_i\alpha$. Použitie pravidiel $A \to B\alpha$ a $B \to \beta_i$ je určené rovnakými k vopred prezeranými symbolmi. Ak vytvárame zodpovedajúcu deriváciu v G_1, tak použitie pravidla $A \to \beta_i\alpha$ je určené rovnakými symbolmi.

Veta 6.9. Ak bezkontextová gramatika $G = (N, T, P, S)$ je LL(1) gramatika, tak existuje q-gramatika $G' = (N, T, P', S)$ taká, že $L(G) = L(G')$.

Transformácia, ktorá upraví LL(1) gramatiku na q-gramatiku, je založená na opakovane vykonávanej rohovej substitúcii. Uveďme teraz algoritmus, ktorý túto transformáciu vykonáva.

Algoritmus 6.11.
Transformácia LL(1) gramatiky na q-gramatiku.
Vstup: LL(1) gramatika $G = (N, T, P, S)$.
Výstup: q-gramatika $G' = (N, T, P', S)$ taká, že $L(G) = L(G')$.
Metóda:

1. Na množine neterminálnych symbolov N zavedieme čiastočné usporiadanie, aby platilo: ak $A_i \overset{*}{\Rightarrow} A_j\alpha$, tak $i < j$. Nech $N = \{A_1, A_2, \ldots, A_n\}$. Môžeme to urobiť, pretože gramatika G je LL(1) gramatikou, a preto nemôže byť zľava rekurzívna (pozri vetu 6.3). Vidieť, že pravé strany pravidiel so symbolom A_n na ľavej strane musia začínať terminálnymi symbolmi, alebo sú to prázdne reťazce.
2. Položme $i = n - 1$ a $P' = P$.
3. Ak $i = 0$, získali sme gramatiku G' a algoritmus končí, inak pokračujeme krokom 4.
4. Každé pravidlo tvau $A_i \to A_j\alpha$, kde $j > i$ v P', nahradíme pravidlami $A_i \to \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_m\alpha$, pričom $A_j \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ sú všetky pravidlá v P' s neterminálnym symbolom A_j na ľavej strane.
5. Ak všetky pridané pravidlá majú pravé strany začínajúce terminálnymi symbolmi, alebo sú tvorené prázdnymi reťazcami, pokračujeme krokom 6, inak krokom 4.
6. Polož $i = i - 1$ a pokračuj krokom 3.

Gramatika G' je ekvivalentná s gramatikou G, pretože bola použitá iba substitúcia podľa vety 6.8. Gramatika G' je LL(1) pretože gramatika G bola LL(1) a substitúcia túto vlastnosť zachováva. Zostáva ešte ukázať, že G' je q-gramatika, to znamená, že pravá strana každého pravidla začína alebo terminálnym symbolom, alebo je tvorená prázdnym reťazcom. To je ale zrejmé, pretože krok 4 sa aplikuje na všetky pravidlá, ktorých pravé strany začínajú neterminálnym symbolom, a to tak na pravidlá, ktoré boli v P, ako aj na pravidlá, ktoré vznikli substitúciou.

Príklad 6.24. Daná je LL(1) gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

$S \to cS$ $\qquad A \to aS$
$S \to Ac$ $\qquad A \to b$

Upravíme túto gramatiku na q-gramatiku.

1. Zavedieme usporiadanie $N = \{S, A\}$, t. j. $A_1 = S$, $A_2 = A$.
2. Položíme $i = 1$ a $P' = P$.
4. Pravidlo $S \to Ac$ v P' nahradíme pravidlami

$S \to aSc$
$S \to bc$

6. $i = 0$.
3. Algoritmus končí a dostávame gramatiku

$G' = (\{S, A\}, \{a, b, c\}, P', S)$

kde P' obsahuje pravidlá

$S \to cS \mid aSc \mid bc$
$A \to aS \mid b$

Teraz vidíme, že symbol A je nedostupný, a preto ho môžeme z gramatiky vylúčiť. Po tejto úprave dostaneme gramatiku $G_1 = (\{S\}, \{a, b, c\}, P_1, S)$, kde P_1 obsahuje pravidlá $S \to cS \mid aSc \mid bc$

Príklad 6.25. Daná je LL(1) gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

$S \to AB$
$A \to aA \mid e$
$B \to bA \mid e$

Upravme túto gramatiku na q-gramatiku.
1. Zavedieme usporiadanie $N = \{S, A, B\}$.
2. Položíme $i = 2$ a $P' = P$.
4. Neurobíme nič.
6. $i = 1$.
4. Pravidlo $S \to AB$ v P' nahradíme pravidlami $S \to aAB \mid B$.
5. Krok 4 treba opakovať.
4. Pravidlo $S \to B$ v P' nahradíme pravidlami $S \to bA \mid e$.
3. Algoritmus končí a dostávame gramatiku $G' = (\{S, A, B\}. \{a, b\}, P', S)$, kde P' obsahuje pravidlá

$S \to aAB \mid bA \mid e$
$A \to aA \mid e$
$B \to bA \mid e$

Poznámka 6.4. Výhodou úpravy LL(1) gramtiky na q-gramatiku je zrýchlenie syntaktickej analýzy. Postupnou substitúciou sa dosiahne to, že sa zmenší počet krokov pri analýze (pozri príklad 6.26). Niekedy môže byť nevýhodou, že vytvorená q-gramatika má väčší počet pravidiel ako pôvodná LL(1) gramatika. V žiadnom prípade sa však nezväčší počet neterminálnych symbolov.

Ten sa dokonca v niektorých prípadoch môže zmenšiť (pozri príklad 6.24). Z uvedených skutočností vyplýva, že niekedy je lepšie použiť na konštrukciu syntaktického analyzátora LL(1) gramatiku, inokedy ekvivalentnú q-gramatiku.

Príklad 6.26. V gramatike G z príkladu 6.25 má veta aab túto ľavú deriváciu: $S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aabA \Rightarrow aab$.

V gramatike G' z príkladu 6.25 má tá istá veta aab túto ľavú deriváciu: $S \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aabA \Rightarrow aab$.

Derivácia v gramatike G' je o jeden krok kratšia, čo znamená, že aj syntaktická analýza sa vykonáva rýchlejšie.

Druhá transformácia, ktorú tu uvedieme, umožňuje transformovať slabú LL(k) gramatiku na silnú LL(k) gramatiku. Táto transformácia úzko súvisí s konštrukciou syntaktického analyzátora pre slabé LL(k) gramatiky. Ak chceme pri syntaktickej analýze slabých LL(k) gramatík používať algoritmus pre silné LL(k) gramatiky, musíme rozkladovú tabuľku upraviť tak, že neterminálne symboly na pravých stranách pravidiel v rozkladovej tabuľke nahradíme zodpovedajúcimi stavmi príslušného LL automatu. Transformácia slabej LL(k) gramatiky na silnú potom spočíva v tom, že tieto stavy považujeme za nové neterminálne symboly, ktoré generujú rovnaké konštrukcie ako pôvodné neterminálne symboly.

Algoritmus 6.12.

Transformácia slabej LL(k) gramatiky na silnú LL(k) gramatiku.
Vstup: slabá LL(k) gramatika $G = (N, T, P, S)$.
Výstup: silná LL(k) gramatika $G' = (N', T, P', S)$ taká, že $L(G) = L(G')$.
Metóda:
1. Vytvoríme súbor $\mathscr{L}$ množín LL(k) položiek pre G.
2. Položíme $P_1 = \{A \to \alpha / \alpha \in T^*, A \to \alpha \in P\}$ a $N_1 = \mathscr{L} \cup \{S'\}$.
3. Pre každú množinu $D_i \in \mathscr{L}$ vykonáme: Ak je v množine D_i súboru $\mathscr{L}$ položka tvaru $[B \to \alpha.A\beta, u]$, kde $A \in N$, tak vložíme do P_1 pravidlo tvaru $B \to \alpha D_i \beta$.
4. Pre každý neterminálny symbol D_i, ktorým bol nahradený neterminálny symbol A na pravej strane niektorého pravidla podľa bodu 3 , vytvoríme pravidlá tvaru $D_i \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$, ak v P' sú pravidlá $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$.
5. Vylúčením nadbytočných symbolov z gramatiky $G_1 = (N_1, T, P_1, S)$ dostaneme výslednú gramatiku $G' = (N', T, P', S)$.

Príklad 6.27. Gramatiku $G = (\{S, S', A\}, \{a, b\}, P, S')$, kde P obsahuje pravidlá

$S' \to S$
$S \to aAaa \mid bAba$
$A \to b \mid e$

budeme transformovať na silnú LL(2) gramatiku.

1. Súbor množín LL(2) položiek bol vytvorený v príklade 6.17.
2. $P_1 = \{A \rightarrow b \mid e\}$, $N_1 = \{D_i / i = 0, 1, \ldots, 10\} \cup \{S'\}$.
3. Do P_1 pridáme postupne pravidlá

$$S' \rightarrow D_1$$
$$S \rightarrow aD_5aa$$
$$S \rightarrow bD_7aa$$

4. Do P_1 pridáme pravidlá

$$D_1 \rightarrow aD_5aa$$
$$D_1 \rightarrow bD_7ba$$
$$D_5 \rightarrow b \mid e$$
$$D_7 \rightarrow b \mid e$$

5. Po vylúčení nadbytočných symbolov dostaneme gramatiku $G' = (\{S', D_1, D_5, D_7\}, \{a, b\}, P', S')$, kde P' obsahuje pravidlá

$$S' \rightarrow D_1$$
$$D_1 \rightarrow aD_5aa \mid bD_7ba$$
$$D_5 \rightarrow b \mid e$$
$$D_7 \rightarrow b \mid e$$

Táto gramatika je silná LL(2) gramatika.

6.1.5 Transformácie bezkontextových gramatík na LL(k) gramatiky

Je známe, že vo všeobecnosti nemožno nájsť algoritmus, ktorý by ľubovoľnú bezkontextovú gramatiku transformoval na LL(k) gramatiku. Dôvodom je skutočnosť, že pre niektoré bezkontextové jazyky neexistujú LL(k) gramatiky. Príkladom môže byť jazyk $L = \{a^n b^n / n \geqq 0\} \cup \{a^n c^n / n \geqq 0\}$ z príkladu 6.19.

Ďalším problémom je skutočnosť, že neexistuje transformačná metóda, ktorá vedie k cieľu, ani v prípade, že transformujeme bezkontextovú gramatiku, ku ktorej existuje ekvivalentná LL(k) gramatika. Napriek tomu môžeme v mnohých prípadoch jednoduchými transformáciami získať z bezkontextovej gramatiky LL(k) gramatiku.

Základné transformácie, ktoré možno použiť na získanie ekvivalentnej LL(1) gramatiky k danej bezkontextovej gramatike, môžeme zhrnúť takto:

1. Odstránenie ľavej rekurzie.
2. Ľavá faktorizácia.
3. Rohová substitúcia
4. Pohltenie terminálneho symbolu.

5. Extrakcia pravého kontextu.
6. Pohltenie reťazca.

Odstránenie ľavej rekurzie z gramatiky opisuje algoritmus 4.7. Substitúcia bola opísaná v odseku 6.1.4 (pozri vetu 6.8). Ostatnými transformáciami sa budeme zaoberať teraz.

Ľavá faktorizácia je transformácia, ktorú opisuje nasledujúca veta.

Veta 6.10. Nech $G = (N, T, P, S)$ je bezkontextová gramatika, kde v P sú obsiahnuté pravidlá $A \rightarrow \alpha\alpha_1 \mid \alpha\alpha_2 \mid \ldots \mid \alpha\alpha_n$, a pritom $\alpha \neq e$. Potom existuje gramatika $G' = (N \cup \{A'\}, T, P', S)$, kde $P' = P - \{A \rightarrow \alpha\alpha_1 \mid \alpha\alpha_2 \mid \ldots \mid \alpha\alpha_n\} \cup \cup \{A \rightarrow \alpha A'\} \cup \{A' \rightarrow \alpha_1 \mid \alpha_2 \ldots \mid \alpha_n\}$ taká, že $L(G) = L(G')$.

Na dôkaz ekvivalencie gramatík G a G' stačí dosadiť v G' do pravidla $A \rightarrow \alpha A'$ za A' pravidlá $A' \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ a dostaneme gramatiku G. Táto substitúcia zachováva jazyk generovaný pôvodnou gramatikou.

Operáciu pohltenia terminálu opisuje nasledujúca veta.

Veta 6.11. Nech $G = (N, T, P, S)$ je bezkontextová gramatika, ktorá obsahuje pravidlo $A \rightarrow \alpha Ba\beta$, $a \in T$, $B \in N$, α, $\beta \in (N \cup T)^*$ a pravidlá $B \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m$. Potom existuje bezkontextová gramatika $G' = (N \cup \{[Ba]\}, T, P', S)$, kde $P' = P - \{A \rightarrow \alpha Ba\beta\} \cup \{A \rightarrow \alpha[Ba]\beta\} \cup \{[Ba] \rightarrow \alpha_1 a \mid \alpha_2 a \mid \ldots \mid \alpha_m a$ taká, že $L(G) = L(G')$.

Ak dosadíme do pravidla $A \rightarrow \alpha[Ba]\beta$ v gramatike G' za neterminálny symbol $[Ba]$, dostaneme gramatiku G'', ktorú by sme inak získali dosadením za B do pravidla $A \rightarrow \alpha Ba\beta$ v gramatike G. Podľa vety o substitúcii platí

$$L(G'') = L(G'),\ L(G'') = L(G) \Rightarrow L(G) = L(G')$$

Extrakcia pravého kontextu je operácia, ktorú opisuje nasledujúca veta.

Veta 6.12 Nech $G = (N, T, P, S)$ je bezkontextová gramatika bez ľavej rekurzie, ktorá obsahuje pravidlo $X \rightarrow \alpha AY\beta$, kde $a \in \mathrm{FIRST}(Y\beta)$, $X, A, Y \in N$, α, $\beta \in (N \cup T)^*$,

$$Y\beta \overset{*}{\Rightarrow} a\gamma_1,\ Y\beta \overset{*}{\Rightarrow} a\gamma_2,\ \ldots,\ Y\beta \overset{*}{\Rightarrow} a\gamma_m$$
$$Y\beta \overset{*}{\Rightarrow} \delta_1,\ Y\beta \overset{*}{\Rightarrow} \delta_2,\ \ldots,\ Y\beta \overset{*}{\Rightarrow} \delta_n$$

a platí, že $a \notin \mathrm{FIRST}(\delta_i)$ pre $1 \leqq i \leqq n$ a pritom nie je v tejto gramatike možná derivácia $Y\beta \overset{*}{\Rightarrow} X\gamma$.

Potom existuje gramatika $G' = (N, T, P', S)$, kde

$$P' = P - \{X \rightarrow \alpha AY\beta\} \cup \{X \rightarrow \alpha Aa\gamma_1 \mid \alpha Aa\gamma_2 \mid \ldots \mid \alpha Aa\gamma_m \mid \alpha A\delta_1 \mid \alpha A\delta_2 \mid \ldots \mid \alpha A\delta_n\}$$

taká, že $L(G) = L(G')$.

Gramatiku G' získame z G postupným dosadzovaním za neterminálne symboly, ktoré sa vyskytujú za symbolom A v pravidlách gramatiky G. Za predpokladu, že v gramatike G nie je ľavá rekurzia a neplatí, že $Y\beta \overset{*}{\Rightarrow} X\gamma$, vedie

tento postup k cieľu, pretože po konečnom počte dosadení sa za symbolom A vyskytnú terminálne symboly. Ekvivalencia gramatík G a G' je zrejmá.

Operáciu pohltenia reťazca opisuje nasledujúca veta.

Veta 6.13. Nech $G = (N, T, P, S)$ je bezkontextová gramatika, ktorá obsahuje pravidlá $A \to \alpha B\beta\gamma$, kde $B \in N$, α, β, $\gamma \in (N \cup T)^*$, $B \to \alpha_1 | \alpha_2 | \ldots | \alpha_m$. Potom existuje bezkontextová gramtika $G' = (N \cup \{[B\beta\}, T, P', S)$, kde

$$P' = P - \{A \to \alpha B\beta\gamma\} \cup \{A \to \alpha[B\beta\gamma\} \cup \{[B\beta \to \alpha_1\beta | \alpha_2\beta | \ldots | \alpha_m\beta\}$$

taká, že $L(G) = L(G')$.

Do pravidla $A \to \alpha[B\beta\gamma$ dosadíme za $[B\beta]$ a dostaneme G'' takú, že $L(G') = L(G'')$. Ďalej dosadíme za B v gramatike G a dostaneme znova G'' a platí $L(G) = L(G'')$. Preto platí aj $L(G) = L(G')$.

Teraz ukážeme, akým spôsobom možno v niektorých prípadoch transformovať bezkontextovú gramatiku na LL(1) gramatiku. Podľa vety 6.6 neexistuje všeobecný algoritmus, ktorým by bolo možné urobiť takúto transformáciu. To znamená, že žiadna z uvedených transformácií nemusí viesť k LL(1) gramatike. Uvidíme však, že v niektorých prípadoch LL(1) gramatiku získame. V prípade, že niektoré transformácie budú končiť neúspechom, nemožno ešte povedať, že ekvivalentná LL(1) gramatika neexistuje. Dá sa povedať iba toľko, že danou transformáciou LL(1) gramatiku získať nemožno.

Dôvody, prečo bezkontextová gramatika nie je LL(1) gramatikou, možno zhrnúť do týchto dvoch bodov:

1. Definícia gramatiky vyžaduje, aby pre každú dvojicu pravidiel $A \to \beta$ a $A \to \gamma$ platilo FIRST $(\beta) \cap$ FIRST $(\gamma) = \emptyset$. Túto podmienku budeme nazývať *podmienka* FIRST-FIRST a jej nesplnenie *kolízia* FIRST-FIRST.
2. Definícia LL(1) gramatiky ďalej vyžaduje, aby pre každú dvojicu pravidiel $A \to \beta$ a $A \to \gamma$ platilo FOLLOW$(A) \cap$ FIRST$(\gamma) = \emptyset$ v prípade, že z reťazca β možno derivovať prázdny reťazec. Túto podmienku budeme nazývať *podmienka* FIRST-FOLLOW a jej nesplnenie *kolízia* FIRST-FOLLOW.

Najjednoduchším prípadom kolízie FIRST-FIRST je prítomnosť pravidiel tvaru $A \to \alpha\alpha_1 | \alpha\alpha_2 | \ldots | \alpha\alpha_n$. Na odstránenie tejto kolízie možno použiť ľavú faktorizáciu.

Príklad 6.28. Daná je bezkontextová gramatika $G = (\{L, S\}, \{a, ;\}, P, L)$, kde P obsahuje pravidlá

$L \to S | S; L$
$S \to a$

Pretože FIRST $(S) \cap$ FIRST $(S;\ L) = \{a\}$, gramatika G nie je LL(1) gramatikou. Ak urobíme ľavú faktorizáciu, dostaneme gramatiku $G' = (\{L, S, L'\}, \{a, ;\}, P', L)$, kde P' obsahuje pravidlá

$L \to SL'$
$L' \to e | ; L$
$S \to a$

Táto gramatika je už LL(1).

Ľavá faktorizácia však nezaručuje, že výsledná gramatika bude LL(1).

Príklad 6.29. Daná je gramatika $G = (\{A, B, C\}, \{a, x, y, z\}, P, A)$, kde P obsahuje pravidlá

$A \to aBxx | aCyy | zy | zx$
$B \to aBx | z$
$C \to aCy | z$

Po ľavej faktorizácii dostaneme gramatiku $G' = (\{A, A', A'', B, C\}, \{a, x, y, z\}, P', A)$, kde P' obsahuje pravidlá

$A \to aA' | zA''$
$A' \to Bxx | Cyy$
$A'' \to x | y$
$B \to aBx | z$
$C \to yCy | z$

Táto gramatika nie je LL(1) gramatikou, rovnako ako gramatika G, pretože

FIRST $(Bxx) \cap$ FIRST $(Cyy) = \{z\}$

Je zaujímavé, že bola nájdená trieda gramatík, pre ktoré je možné ľavou faktorizáciou získať LL(k) gramatiky [11]. Sú to LP(k) gramatiky (left part), ktoré sú definované takto:

Definícia 6.15. Bezkontextová gramatika $G = (N, T, P, S)$ je *LP(k) gramatikou*, ak pre každé dve rôzne pravidlá v P tvaru

$A \to \alpha\beta_1$ a $A \to \alpha\beta_2$

v ktorých α je najdlhšia spoločná predpona, z existencie derivácií

$S \overset{*}{\Rightarrow} wA\gamma \Rightarrow w\beta_1\gamma$
$S \overset{*}{\Rightarrow} wA\gamma/' \Rightarrow w\beta_2\gamma/'$

vyplýva, že FIRST$_k(\beta_1\gamma) \cap$ FIRST$_k(\beta_2\gamma/') = \emptyset$.

Silné LP(k) gramatiky sú definované takto:

Ak pre každé dve rôzne pravidlá v P tvaru $A \to \alpha\beta_1$, $A \to \alpha\beta_2$ v ktorých je α najdlhšia spoločná predpona, platí:

FIRST$_k(\beta_1$FOLLOW$_k(A)) \cap$ FIRST$_k(\beta_2$FOLLOW$_k(A)) = \emptyset$

Bezkontextová gramatika G sa potom nazýva *silná LP(k) gramatika.*

LP(k) gramatika sa dá ľavou faktorizáciou transformovať na LL(k) gramatiku. Podobne silná LP(k) gramatika sa dá ľavou faktorizáciou transformovať na silnú LL(k) gramatiku.

Niekedy sa v gramatike vyskytujú pravidlá tvaru

$$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

ktoré obsahujú kolíziu FIRST-FIRST, no nemožno ich priamo faktorizovať. V takomto prípade môžeme použiť rohovú substitúciu a upraviť tieto pravidlá na tvar, keď je možné urobiť ľavú faktorizáciu.

Príklad 6.30. Daná je bezkontextova gramatika $G = (\{A, B, C\}, \{a, b, c, d\}, P, A)$, kde P obsahuje pravidlá

$$A \to aB \mid CB$$
$$C \to aC \mid bB$$
$$B \to cB \mid d$$

Táto gramatika nie LL(1), pretože FIRST $(aB) \cap$ FIRST $(CB) = \{a\}$. Pritom však nemožno urobiť priamo ľavú faktorizáciu. Urobíme preto najskôr rohovú substitúciu v pravidle $A \to CB$ a dostaneme gramatiku s pravidlami

$$A \to aB \mid aCB \mid bBB$$
$$B \to cB \mid d$$
$$C \to aC \mid bB$$

Teraz už možno urobiť ľavú faktorizáciu tak, že pravidlá $A \to aB \mid aCB$ nahradíme pravidlami

$$A \to aA'$$
$$A' \to B \mid CB$$

Výsledná gramatika je gramatika

$G' = (\{A, A', B, B', C\}, \{a, b, c, d\}, P', A)$, kde P' obsahuje pravidlá

$$A \to aA'$$
$$A' \to B \mid CB$$
$$B \to cB \mid d$$
$$C \to aC \mid bB$$

Táto gramatika je už LL(1).

Ak v množine pravidiel s rovnakou ľavou stranou, ktorá obsahuje kolíziu FIRST-FIRST, je viac takých pravidiel, ktorých pravé strany začínajú neterminálnymi symbolmi, je možné rôznou voľbou poradia substitúcie ovplyvniť počet substitúcií, a tým aj počet zavedených pravidiel.

Príklad 6.31. Daná je gramatika $G = (\{A, B, D\}, \{b, c, d, x, y, z\}, P, A)$, kde P obsahuje pravidlá

$$A \to Bc \mid Dd$$
$$B \to bx \mid y$$
$$D \to Bz$$

Pretože FIRST $(Bc) \cap$ FIRST $(Dd) = \{b, y\}$, gramatika G nie je LL(1). Pretože nemožno urobiť priamo ľavú faktorizáciu, urobíme substitúciu za B v pravidle $A \to Bc$. Po tejto substitúcii dostaneme pre symbol A pravidlá

$$A \to bxc \mid yc \mid Dd$$

Pretože ani teraz nemôžeme urobiť novú faktorizáciu, musíme urobiť ďalšiu substitúciu za D v pravidle $A \to Dd$. Po tejto substitúcii dostaneme pre symbol A pravidlá

$$A \to bxc \mid yc \mid Bzd$$

Ani teraz ešte nemožno urobiť ľavú faktorizáciu. Urobíme ďalšiu substitúciu v pravidle $A \to Bzd$. Teraz dostaneme pre symbol A pravidlá

$$A \to bxc \mid yc \mid bxzd \mid yzd$$

a môžeme urobiť ľavú faktorizáciu. Dostaneme pravidlá

$$A \to bxA' \mid yA''$$
$$A' \to c \mid zd$$
$$A'' \to c \mid zd$$

Pretože pravidlá pre A' a A'' sú rovnaké, nahradíme symbol A'' symbolom A' a pravidlá pre A'' vynecháme. Dostaneme gramatiku $G' = (\{A, A'\}, \{b, c, d, x, y, z\}, P, A)$, kde P obsahuje pravidlá

$$A \to bxA' \mid yA'$$
$$A' \to c \mid zd$$

Táto gramatika je LL(1) gramatika.

Podobný výsledok dosiahneme aj substitúciou v pravidle $A \to Dd$ za symbol D v pôvodnej gramatike. Po tejto substitúcii dostaneme pre symbol A pravidlá $A \to Bc \mid Bzd$ a môžeme okamžite urobiť ľavú faktorizáciu. Dostaneme pravidlá

$$A \to BA'$$
$$A_2 \to c \mid zd$$

Výsledná gramatika je $G'' = (\{A, A', B\}, \{b, c, d, x, y, z\}, P'', A)$, kde P'' obsahuje pravidlá

$A \rightarrow BA'$
$A' \rightarrow c \mid zd$
$B \rightarrow bx \mid y$

Táto gramatika je tiež LL(1) gramatika.

Príčinou toho, že substitúcia v pravidle $A \rightarrow Dd$ vedie rýchlejšie k tvaru pravidiel vhodnému na ľavú faktorizáciu než substitúcia v pravidle $A \rightarrow Bc$, je skutočnosť, že v uvedenej gramatike platí $D \stackrel{*}{\Rightarrow} Ba$. Túto skutočnosť možno využiť pri výbere pravidiel na substitúciu. Uvedená metóda, t.j. rohová substitúcia spolu s ľavou faktorizáciou, nezaručuje vo všeobecnosti, že výsledná gramatika bude LL(1). Ukážeme si to v nasledujúcom príklade.

Príklad 6.32. Daná je gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

$S \rightarrow A \mid B$
$A \rightarrow cA \mid a$
$B \rightarrow cB \mid b$

Pretože FIRST $(A) \cap$ FIRST $(B) = \{c\}$, nie je táto gramatika LL(1). Urobíme preto substitúciu v pravidlách $S \rightarrow A \mid B$. Dostaneme pravidlá $S \rightarrow a \mid b \mid cA \mid cB$. Po ľavej faktorizácii získame novú množinu pravidiel:

$S \rightarrow a \mid b \mid cS'$
$S' \rightarrow A \mid B$
$A \rightarrow cA \mid a$
$B \rightarrow cB \mid b$

Po ľavej faktorizácii majú pravidlá $S' \rightarrow A \mid B$ úplne rovnaký charakter ako pravidlá $S \rightarrow A \mid B$, ktorých odstránenie bolo dôvodom k substitúcii a faktorizácii. To znamená, že gramatiku G nemožno touto metódou transformovať na LL(1) gramatiku. Pritom však jazyk

$$L(G) = \{c^n a / n \geqq 0\} \cup \{c^n b / n \geqq 0\}$$

je LL(1) jazyk, ktorý možno generovať napríklad LL(1) gramatikou $G' = (\{S, A, C\}, \{a, b, c\}, P', S)$, kde P' obsahuje pravidlá

$S \rightarrow CA$
$A \rightarrow a \mid b$
$C \rightarrow cC \mid e$

Teraz sa budeme zaoberať odstraňovaním kolízie FIRST-FOLLOW. Uvedieme príklad gramatiky, ktorá obsahuje kolíziu FIRST-FOLLOW. Túto kolíziu odstránime pohltením terminálneho symbolu, ktorý kolíziu spôsobuje.

Príklad 6.33. Daná je bezkontextová gramatika $G = (\{A, B, C\}, \{a, b, c\}, P, A)$, kde P obsahuje pravidlá

$A \rightarrow BaC$
$B \rightarrow e \mid aaC$
$C \rightarrow c \mid bC$

V tejto gramatike je kolízia FIRST-FOLLOW, pretože

FIRST $(aaC) \cap$ FOLLOW $(B) = \{a\}$

Túto kolíziu odstránime pohltením terminálneho symbolu a za symbolom B. Zavedieme nový neterminálny symbol, ktorý označíme $[Ba]$ a vytvoríme pravidlá $[Ba] \rightarrow a \mid aaCa$. Ďalej nahradíme v pravidle $A \rightarrow BaC$ reťazec Ba symbolom $[Ba]$. Dostaneme gramatiku s pravidlami

$A \rightarrow [Ba]C$
$B \rightarrow e \mid aaC$
$C \rightarrow c \mid bC$
$[Ba] \rightarrow a \mid aaCa$

Po úprave (vylúčení nedostupných symbolov a ľavej faktorizácii) dostaneme gramatiku $G' = (\{A, C, [Ba], [Ba]'\}, \{a, b, c\}, P', A)$ kde P' obsahuje pravidlá

$A \rightarrow [Ba]C$ $\qquad C \rightarrow c \mid bC$
$[Ba] \rightarrow a[Ba]'$ $\qquad [Ba]' \rightarrow e \mid aCa$

Táto gramatika je LL(1).

Príklad 6.34. Daná je bezkontextová gramatika $G = (\{A, B, C\}, \{a, b, c\}, P, A)$, kde P obsahuje pravidlá

$A \rightarrow BaC$
$B \rightarrow e \mid abC$
$C \rightarrow c \mid bC$

V tejto gramatike je kolízia FIRST-FOLLOW, rovnako ako v predchádzajúcom prípade. Rovnakým postupom získame gramatiku $G' = (\{A, C, [Ba], [Ba]'\}, \{a, b, c\}, P', A)$, kde P' obsahuje pravidlá

$A \rightarrow [Ba]C$
$C \rightarrow c \mid bC$
$[Ba] \rightarrow a[Ba]'$
$[Ba]' \rightarrow e \mid bCa$

Gramatika G' obsahuje znova kolíziu FIRST-FOLLOW, pretože platí FIRST $(bCa) \cap$ FOLLOW $([Ba]') = \{b\}$. Ak by sme týmto spôsobom pokračovali

ďalej, zistíme, že odstránenie jednej kolízie FIRST-FOLLOW má za následok zavedenie ďalšej kolízie FIRST-FOLLOW. Pritom jazyk $L(G_1) = \{ab^n cab^m c / n \geqq 1, m \geqq 0\} \cup \{ab^m c / m \geqq 0\}$ je možné generovať LL(1) gramatikou $G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

$S \to aA$
$A \to c \mid bB$
$B \to bB \mid cC$
$C \to e \mid aD$
$D \to bD \mid c$

Pohltenie terminálneho symbolu možno urobiť priamo iba vtedy, ak sa príslušný terminálny symbol vyskytuje v pravidle bezprostredne za neterminálnym symbolom. Pri redukcii množiny FOLLOW(A) môže nastať situácia, keď sa terminálny symbol, ktorý chceme vylúčiť z FOLLOW(A), nevyskytuje bezprostredne za symbolom A v niektorom pravidle, ale situácia je podobná nasledujúcej: V gramatike existuje pravidlo $X \to \alpha AY\beta$, a pritom platí, že $Y\beta \overset{*}{\Rightarrow} a\gamma$. V takomto prípade musíme gramatiku upraviť tak, aby sa symbol a vyskytol v pravidle $X \to \alpha Aa\gamma$ priamo za symbolom A. To je možné v niektorých prípadoch dosiahnuť extrakciou pravého kontextu.

Príklad 6.35. Daná je gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

$S \to AB \mid cB$
$A \to aAcB \mid e$
$B \to ab$

Táto gramatika nie je LL(1), pretože platí FIRST$(aAcB) \cap$ FOLLOW$(A) = \{a\}$. Aby sme túto gramatiku mohli transformovať na LL(1) gramatiku, musíme zredukovať množinu FOLLOW(A) o symbol a. Skutočnosť, že symbol a je prvkom množiny FOLLOW(A), je spôsobená pravidlom $S \to AB$. Pretože za symbolom A nie je priamo terminálny symbol a, ale neterminálny symbol B, pre ktorý platí $a \in$ FIRST(B), musíme najprv urobiť extrakciu pravého kontextu symbolu A v pravidle $S \to AB$ a až potom urobiť pohltenie terminálu. Po extrakcii pravého kontextu dostaneme gramatiku s pravidlami

$S \to Aab \mid cB$
$A \to aAcB \mid e$
$B \to ab$

Teraz urobíme pohltenie symbolu a v pravidle $S \to Aab$. Výsledkom bude gramatika $G_1 = (\{S, A, B, [Aa]\}, \{a, b, c\}, P_1, S)$ s pravidlami

$S \to [Aa]b \mid cB$ $\qquad B \to ab$
$A \to aAcB \mid e$ $\qquad [Aa] \to aAcBa \mid a$

Po ľavej faktorizácii pravidiel pre $[Aa]$ dostaneme LL(1) gramatiku $G' = (\{S, A, B, [Aa], [Aa]'\}, \{a, b, c\}, P', S)$ s pravidlami

$S \to [Aa]b \mid cB$ $\qquad [Aa] \to a[Aa]'$
$A \to aAcB \mid e$ $\qquad [Aa]' \to AcBa \mid e$
$B \to ab$

Extrakciu pravého kontextu nemôžno urobiť vo všetkých prípadoch. Ak máme pravidlá tvaru

$S \to aAS$
$A \to e \mid a$

a urobíme pokus o extrakciu pravého kontextu symbolu A v pravidle $S \to aAS$, dostaneme pravidlo $S \to aAaAS$. V tomto pravidle je pravý kontext symbolu A extrahovaný, ale pri extrakcii sa na pravej strane objavil znova symbol A, za ktorým nasleduje symbol S, čo je rovnaká kombinácia ako v pravidle $S \to aAS$ pred extrakciou pravého kontextu. Uvedený postup v danom prípade vedie k nekonečnému cyklu. Ak pohltenie terminálu pri odstraňovaní kolízie FIRST-FOLLOW nevedie k cieľu, možno použiť operáciu pohltenia reťazca. Nasledujúci príklad ukazuje gramatiku, ktorú nemožno transformovať na LL(1) gramatiku pohltením terminálu, ale táto transformácia je možná pohltením reťazca.

Príklad 6.36. Daná je bezkontextová gramatika $G = (\{A, B, C\}, \{a, b, c\}, P, A)$, kde P obsahuje pravidlá

$A \to BaC$
$B \to e \mid aC$
$C \to c \mid bC$

Táto gramatika nie je LL(1) gramatikou, pretože platí

FIRST$(aC) \cap$ FOLLOW$(B) = \{a\}$

Uvedenú kolíziu FIRST-FOLLOW nemožno odstrániť pohltením terminálu a v pravidle $A \to BaC$. Po pohltení terminálu dostaneme pravidlá

$A \to [Ba]C$ $\qquad C \to c \mid bC$
$B \to e \mid aC$ $\qquad [Ba] \to a \mid aCa$

Po ľavej faktorizácii pravidiel pre symbol $[Ba]$ dostaneme gramatiku $G_1 = (\{A, B, C, [Ba], [Ba]'\}, \{a, b, c\}, P_1, A)$, kde P_1 obsahuje pravidlá

$A \to [Ba]C$ $\qquad [Ba] \to a[Ba]'$
$B \to e \mid aC$ $\qquad [Ba]' \to e \mid C$
$C \to c \mid bC$

Táto gramatika nie je LL(1) gramatikou, pretože

FIRST (C) $\cap$ FOLLOW ($[Ba]'$) = $\{b, c\}$

To znamená, že použitá metóda neviedla k cieľu. Ak použijeme na transformáciu gramatiky G na LL(1) gramatiku operáciu pohltenia reťazca v pravidle $A \to BaC$, dostaneme gramatiku s pravidlami

$A \to [BaC]$ $\quad$ $C \to c \mid bC$
$B \to e \mid aC$ $\quad$ $[BaC] \to aCaC \mid aC$

Po ľavej faktorizácii pravidiel pre $[BaC]$ dostaneme LL(1) gramatiku $G' = (\{A, C, [BaC], [BaC]'\}, \{a, b, c\}, P', A)$, kde P' obsahuje pravidlá

$A \to [BaC]$ $\quad$ $[BaC] \to aC[BaC]'$
$C \to c \mid bC$ $\quad$ $[BaC]' \to aC \mid e$

Použitie operácie pohltenia reťazca pri odstraňovaní kolízie FIRST-FOLLOW má tiež svoje hranice, ako možno vidieť aj z nasledujúceho príkladu.

Príklad 6.37. Daná je gramatika $G = (\{S, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

$S \to aSaB \mid c$
$B \to e \mid ab$

V tejto gramatike je kolízia FIRST-FOLLOW pre pravidlá $B \to e \mid ab$. Po pohltení reťazca aB dostaneme gramatiku s pravidlami

$S \to a[SaB] \mid c$
$B \to e \mid ab$
$[SaB] \to a[SaB]aB \mid caB$

Touto transformáciou sme však nezískali LL(1) gramatiku, pretože FIRST (ab) $\cap$ FOLLOW (B) = $\{a\}$. Pritom možno danú gramatiku transformovať na LL(1) gramatiku opakovaným pohltením terminálu.

Cvičenia

1. Daná je gramatika
G = ({⟨blok⟩, ⟨deklarácie⟩, ⟨zvyšok zoznamu deklarácií⟩, ⟨príkaz⟩, ⟨zvyšok zoznamu príkazov⟩}, {**begin**, **end**, ;, p, d/}, P, ⟨blok⟩), kde P obsahuje pravidlá:
1. ⟨blok⟩ → **begin** ⟨deklarácie⟩; ⟨zvyšok zoznamu deklarácií⟩⟨príkaz⟩ ⟨zvyšok zoznamu príkazov⟩ **end**
2. ⟨zvyšok zoznamu deklarácií⟩ → ⟨deklarácie⟩; ⟨zvyšok zoznamu deklarácií⟩
3. ⟨zvyšok zoznamu deklarácií⟩ → e

4. ⟨zvyšok zoznamu príkazov⟩ → ; ⟨príkaz⟩ ⟨zvyšok zoznamu príkazov⟩
5. ⟨zvyšok zoznamu príkazov⟩ → e
6. ⟨deklarácie⟩ → d
7. ⟨príkaz⟩ → p

Vytvorte pre danú LL(1) gramatiku rozkladovú tabuľku a urobte rozklad vety: **begin** d ; p **end**.

2. Daná je gramatika
G = ({⟨deklarácie⟩, ⟨zoznam id⟩, ⟨id⟩, ⟨zvyšok zoznamu id⟩}, {real, integer, id, ;}, P, ⟨deklarácie⟩), kde P obsahuje pravidlá:
1. ⟨deklarácie⟩ → real⟨zoznam id⟩
2. ⟨deklarácie⟩ → integer⟨zoznam id⟩
3. ⟨zoznam id⟩ → ⟨id⟩⟨zvyšok zoznamu id⟩
4. ⟨zvyšok zoznamu id⟩ → ; ⟨id⟩⟨zvyšok zoznamu id⟩
5. ⟨zvyšok zoznamu id⟩ → e
6. ⟨id⟩ → id

Vytvorte pre danú LL(1) gramatiku rozkladovú tabuľku a urobte rozklad vety: real id ; id.

3. Daná je gramatika pre boolovský výraz
$G = (\{E, T, F, S\}, \{\vee, \wedge, \neg, (,), a\}, P, E/)$, kde P obsahuje pravidlá

$E \to E \vee T \mid T$
$T \to T \wedge F \mid F$
$F \to \neg S \mid S$
$S \to (E/) \mid a$

Transformujte danú gramatiku na LL(1) gramatiku a vytvorte pre ňu rozkladovú tabuľku.

4. Daná je gramatika
$G_i = (\{F, T\}, \{\uparrow, (,), a\}, P_i, F)$, $i = 1, 2$, kde P_1 obsahuje pravidlá

$F \to F\uparrow T \mid T$
$T \to (F) \mid a$

a P_2 pravidlá

$F \to T\uparrow F \mid T$
$T \to (F) \mid a$

Transformujte tieto gramatiky na LL(1) gramatiky.

5. Daná je gramatika
$G = (\{P, L, S\}, \{$**begin**, **end**, ;, $s\}, R, P)$, kde R obsahuje pravidlá

$P \to$ **begin** L **end**
L → L ; S | S
S → s

Transformujte túto gramatiku na LL(1) gramatiku a vytvorte pre ňu rozkladovú tabuľku.

6.2 LR GRAMATIKY A JAZYKY

V tomto článku sa budeme zaoberať skupinou algoritmov syntaktickej analýzy, ktoré vytvárajú derivačný strom analyzovaného reťazca smerom zdola nahor. Tieto algoritmy sa nazývajú algoritmy LR analýzy, pretože čítajú vstupný reťazec zľava (angl. left) a vytvárajú pravý (ang. right) rozklad. Pritom využívajú informácie o najbližších k symboloch v doteraz neprečítanej časti vstupného reťazca. Gramatiky, pre ktoré je možné takéto algoritmy vytvoriť, sa nazývajú LR(k) gramatiky.

Základný princíp syntaktickej analýzy pre LR gramatiky môžeme formulovať takto:

Predpokladajme, že $G = (N, T, P, S)$ je jednoznačná gramatika a že reťazec $w = a_1a_2 \ldots a_n$ je veta v jazyku $L(G)$. Potom existuje pravá derivácia $S = \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_m = w$.

Vzhľadom na to, že táto derivácia je pravá, má každá vetná forma γ_i ($i = 1, 2, \ldots, m - 1$) tvar $\gamma_i = \alpha A a_j a_{j+1} \ldots a_n$, kde $A \in N$, $\alpha \in (N \cup T)^*$ a reťazec $a_j a_{j+1} \ldots a_n \in T^*$ je príponou vety w.

Ak $\gamma_{i-1} = \alpha Bz$ a pravidlo $B \to \beta$ je použité v kroku $\gamma_{i-1} \Rightarrow \gamma_i$, alebo $\alpha Bz \Rightarrow \alpha\beta z$, tak základný problém deterministickej syntaktickej analýzy metódou zdola nahor spočíva v nájdení reťazca βz vo vetnej forme γ_i tak, aby vetná forma γ_i mohla byť redukovaná na vetnú formu γ_{i-1}.

Modelom syntaktického analyzátora pracujúceho metódou zdola nahor je zásobníkový automat skonštruovaný v kapitole 5. Tento zásobníkový automat je vo všeobecnosti nedeterministický, a preto ho nemožno priamo použiť ako syntaktický analyzátor. Vráťme sa teraz k tomuto automatu a rozoberme, za akých okolností a akým spôsobom je možné pre danú bezkontextovú gramatiku vytvoriť deterministický zásobníkový automat. Pre danú bezkontextovú gramatiku vytvoríme pomocou konštrukcie z kapitoly 5 zásobníkový automat, ktorého zobrazenie δ je definované takto (predpokladajme, že zásobník má vrch vpravo):

1. $\delta(q, a, e) = \{(q, a)\}$ pre všetky $a \in T$.
2. $\delta(q, e, \alpha) = \{(q, A)/A \to \alpha \in P\}$.
3. $\delta(q, e, \# S) = \{(r, e)\}$.

Operáciu podľa 1 budeme nazývať *presun*, podľa 2 *redukcia* a podľa 3 *prijatie*.

Z uvedenej konštrukcie vyplýva, že výsledný zásobníkový automat je vždy nedeterministický. Dôvod spočíva v tom, že na presun je definované zobrazenie $\delta(q, a, e) = \{(q, a\}$ a na redukciu zobrazenie $\delta(q, e, \alpha) = \{(q, A)\}$. Pritom reťazec e je vždy predponou aj príponou reťazca α. Aby sme teda mohli hovoriť o deterministickom zásobníkovom automate, je potrebné konštrukciu upraviť. Hlavný dôvod, prečo je uvedený zásobníkový automat nedeterministický, spočíva v tom, že operácia presunu sa robí bez ohľadu na obsah zásobníka. Pokúsime sa preto upraviť zásobníkový automat tak, aby rozhodnutie o tom, či sa má urobiť presun alebo redukcia, vykonal automat podľa toho, aký symbol sa vyskytuje na vrchu zásobníka. Ukážeme túto úpravu na príklade.

Príklad 6.40. Daná je bezkontextová gramatika $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidlá:

$S \to Aa$
$A \to bB \mid Ac$
$B \to d$

Pre túto gramatiku môžeme vytvoriť zásobníkový automat takto:

$R = (\{q, r\}, \{a, b, c, d\}, \{S, A, B, a, b, c, d, \#\}, \delta, q, \#, \{r\})$, kde zobrazenie δ je definované nasledujúco:

1. $\delta(q, a, e) = \{(q, a)\}$
   $\delta(q, b, e) = \{(q, b)\}$
   $\delta(q, c, e) = \{(q, c)\}$
   $\delta(q, d, e) = \{(q, d)\}$
2. $\delta(q, e, Aa) = \{(q, S)\}$
   $\delta(q, e, bB) = \{(q, A)\}$
   $\delta(q, e, Ac) = \{(q, A)\}$
   $\delta(q, e, d) = \{(q, B)\}$
3. $\delta(q, e, \# S) = \{(r, e)\}$

Tento zásobníkový automat je nedeterministický práve vplyvom presunov podľa bodu 1. Tieto presuny možno vykonávať v závislosti od obsahu zásobníka takto:

$\delta(q, a, A) = \{(q, Aa)\}$ — symboly a, c sa vyskytujú vo vetnej forme iba za
$\delta(q, c, A) = \{(q, Ac)\}$ symbolom A,
$\delta(q, b, \#) = \{(q, \# b)\}$ — symbol b sa môže vyskytnúť iba na začiatku vetnej formy,
$\delta(q, d, b) = \{(q, bd)\}$ — symbol d sa môže vyskytnúť iba za symbolom b.

Po tejto úprave získame pre danú gramatiku deterministický zásobníkový automat. Uvedený postup však vedie k deterministickému zásobníkovému automatu iba pre obmedzenú triedu gramatík (pre silné LR(0) gramatiky opísané ďalej), ale pre ostatné gramatiky deterministický zásobníkový automat takto nezískame. Podobne ako pri analýze zhora nadol môžeme aj v tomto prípade využívať na rozhodovanie o tom, akú operáciu vykonať, tieto ďalšie informácie:

a) informáciu o doteraz neprečítanej časti vstupného reťazca,
b) informáciu o doterajšom priebehu analýzy.

Gramatiky, pre ktoré stačí pri analýze metódou zdola nahor na deterministické rozhodovanie iba informácia o najbližších k symboloch v doteraz neprečítanej časti vstupného reťazca, nazývame silné LR(k) gramatiky. V ďalších článkoch sa budeme postupne zaoberať dvoma triedami LR(k) gramatík. Najskôr sa budeme zaoberať silnými LR(k) gramatikami. Pre slabé LR(k) gramatiky je pri deterministickej syntaktickej analýze potrebné využívať informácie o doterajšom priebehu analýzy. Pre obe triedy LR gramatík sa používa až na drobné modifikácie ten istý algoritmus syntaktickej analýzy, založený na princípe zásobníkového automatu. Na rozhodovanie o tom, či a akú redukciu v danom okamihu urobiť, sa vo všetkých prípadoch používajú rozkladové tabuľky, v ktorých sú uvedené potrebné informácie. Rozkladové tabuľky vytvárame na základe bezkontextovej gramatiky a spôsob ich konštrukcie sa pre obe triedy LR gramatík navzájom líši. V nasledujúcich článkoch tieto konštrukcie podrobne opíšeme.

6.2.1 Silné LR gramatiky

Silné LR gramatiky sú také, pre ktoré možno vytvoriť syntaktický analyzátor vykonávajúci syntaktickú analýzu metódou zdola nahor, a pritom

— využíva informácie o najbližších k symboloch v doteraz neprečítanej časti vstupného reťazca,

— nevyužíva informácie o histórii analýzy.

Prv než budeme definovať silnú LR(k) gramatiku, zavedieme funkcie BEFORE a EFF.

Definícia 6.17. *Funkcia* BEFORE (X), kde X je neterminálny symbol z gramatiky $G = (N, T, P, S)$, je definovaná takto:

BEFORE $(X) = \{Y/S \overset{*}{\Rightarrow} \alpha YX\beta,\ Y \in (N \cup T)\} \cup \{\#/S \overset{*}{\Rightarrow} X\beta\}$

Funkcia $\mathrm{EFF}_k(\alpha)$, kde reťazec $\alpha \in (\mathrm{N} \cup \mathrm{T})^*$, je definovaná takto:

$$\mathrm{EFF}_k(\alpha) = \{w/w \in \mathrm{FIRST}_k(\alpha) \text{ a existuje pravá derivácia } \alpha \overset{*}{\Rightarrow} \beta \overset{*}{\Rightarrow} wx \text{ taká, že pre } \beta \text{ neplatí } \beta = Awx\}$$

Do množiny $\mathrm{EFF}_k(\alpha)$ patria všetky reťazce z $\mathrm{FIRST}_k(\alpha)$, ktoré boli derivované deriváciou $\alpha \overset{*}{\Rightarrow} \beta \overset{*}{\Rightarrow} wx$ tak, že prvý neterminálny symbol v β nebol nahradený prázdnym reťazcom. (EFF — e-free first).

Teraz budeme definovať silnú LR(k) gramatiku.

Definícia 6.18. Bezkontextovú gramatiku $G = (N, T, P, S)$ nazývame *silná LR(k) gramatika*, ak pre rozšírenú gramatiku $G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$ platí (pre $P' = P \cup \{S' \to S\}$):

1. Ak sú v P' dvojice pravidiel tvaru
   a) $A \to \alpha X,\ B \to \beta X$,
   b) $A \to \alpha X,\ B \to e$ a $X \in$ BEFORE (B),
   c) $A \to e,\ B \to e$ a $X \in$ BEFORE (B), $X \in$ BEFORE (A),
   tak musí platiť $\mathrm{FOLLOW}_k(A) \cap \mathrm{FOLLOW}_k(B) = \emptyset$.
2. Ak sú v P' dvojice pravidiel tvaru
   a) $A \to \alpha X,\ B \to \alpha X\gamma$,
   b) $A \to e,\ B \to \alpha X\gamma$ a $X \in$ BEFORE (A),
   c) $A \to e,\ B \to \gamma$ a $X \in$ BEFORE (A), $X \in$ BEFORE (B),
   tak musí platiť $\mathrm{FOLLOW}_k(A) \cap \mathrm{EFF}_k(\gamma/\mathrm{FOLLOW}_k(B)) = \emptyset$

Podmienka 1 zabezpečuje, že v prípade, keď sa má vykonávať redukcia, je možné rozhodnúť o pravidle, ktorým sa má redukovať na základe vopred prezeraného reťazca s dĺžkou najviac k symbolov.

Podmienka 2 tejto definície zabezpečuje, aby bolo možné rozhodnúť, či vykonať operáciu presunu alebo redukcie.

Podobne ako pri analýze zhora nadol budeme pri analýze zdola nahor používať pri rozhodovaní o ďalšej vykonávanej operácii rozkladovú tabuľku. V tejto tabuľke budú uložené informácie o tom, akú operáciu budeme vykonávať v závislosti od symbolu, ktorý je na vrchu zásobníka a od vopred prezeraného reťazca. Rozkladovú tabuľku pre silnú LR(k) gramatiku možno vytvoriť podľa nasledujúceho algoritmu.

Algoritmus 6.13.

Vytvorenie rozkladovej tabuľky pre silnú LR(k) gramatiku.

Vstup: silná LR(k) gramatika $G = (N, T, P, S)$.

Výstup: rozkladová tabuľka F pre G.

Metóda: rozkladová tabuľka F je definovaná na karteziánskom súčine $(N \cup T \cup \{\#\}) \times T^{*k}$.

1. Ku gramatike G vytvoríme rozšírenú gramatiku

   $G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$

2. Rozkladovú tabuľku F vytvoríme takto:
   a) $F(X, u)$ = redukcia (i), ak $A \to \alpha X$ je i-té pravidlo v P a $u \in \mathrm{FOLLOW}_k(A)$.
   b) $F(X, u)$ = redukcia (i), ak $A \to e$ je i-té pravidlo v P, $X \in$ BEFORE(A) a $u \in \mathrm{FOLLOW}_k(A)$,
   c) $F(S, e)$ = prijatie,
   d) $F(X, u)$ = presun, ak $B \to \beta X\gamma \in P$ a $u \in \mathrm{EFF}_k(\gamma/\mathrm{FOLLOW}_k(B))$,
   e) $F(X, u)$ = chyba v ostatných prípadoch.

Príklad 6.41. Daná je gramatika $G = (\{E, E', T, T', F\}, \{a, +, *, (,)\}, P, E)$, kde P obsahuje tieto pravidlá

1. $E \to E'T$
2. $E' \to E+$
3. $E' \to e$
4. $T \to T'F$
5. $T' \to T*$
6. $T' \to e$
7. $F \to (E)$
8. $F \to a$

Gramatiku rozšírime o pravidlo 0. $S \to E$.

Táto gramatika se silná LR(1) gramatika a môžeme pre ňu vytvoriť rozkladovú tabuľku (*tab. 6.13*), v ktorej sú jednotlivé operácie označené takto: P — presun, R(i) — redukcia (i), A — prijatie, chybové položky sú prázdne.

Tabuľka 6.13

F	a	$+$	$*$	(	)	e
E		P			P	A
E'	R(6)			R(6)		
T		R(1)	P		R(1)	R(1)
T'	P			P		
F		R(4)	R(4)		R(4)	R(4)
a		R(8)	R(8)		R(8)	R(8)
$+$	R(2)			R(2)		
$*$	R(5)			R(5)		
(	R(3)			R(3)		
)		R(7)	R(7)		R(7)	R(7)
#	R(3)			R(3)		

Pri konštrukcii rozkladovej tabuľky sme využili to, že platí: BEFORE$(E') = \{\#, (\}$, BEFORE$(T') = \{E'\}$.

Syntaktickú analýzu silných LR(k) gramatík budeme robiť pomocou nasledujúceho algoritmu.

Algoritmus 6.14.

Syntaktická analýza pre silné LR(k) gramatiky.

Vstup: rozkladová tabuľka F pre gramatiku $G = (N, T, P, S)$, pravidlá gramatiky G a vstupný reťazec $w \in T^*$.

Výstup: pravý rozklad v prípade, že reťazec $w \in L(G)$, inak chybová signalizácia.

Metóda: algoritmus číta symboly zo vstupného reťazca w, využíva zásobník a vytvára reťazec čísel pravidiel, ktoré boli použité pri jednotlivých redukciách. Na začiatku je v zásobníku symbol #. Opakujeme kroky 1 a 2, kým nenastane prijatie alebo chyba. Symbol na vrchu zásobníka označíme X.

1. Určíme vopred prezeraný reťazec dĺžky k a označíme ho u.
2. a) Ak $F(X, u) =$ presun, prečíta sa vstupný symbol a uloží sa do zásobníka.
   b) Ak $F(X, u) =$ redukcia (i), vylúčime zo zásobníka reťazec α, ktorý je na pravej strane i-tého pravidla $A \to \alpha$. Ak v zásobníku nebol reťazec α, signalizuje sa chyba a analýza končí. Inak do zásobníka vložíme symbol A a k výstupnému reťazcu pripojíme číslo pravidla i.
   c) Ak $F(X, e) =$ prijatie, analýza končí a vystupný reťazec je pravý rozklad vstupného reťazca v prípade, že v zásobníku je reťazec $\# X$, inak sa vykoná chybová signalizácia.
   d) Ak $F(X, u) =$ chyba, analýza končí chybovou signalizáciou.

Konfiguráciou algoritmu budeme rozumieť trojicu (α, x, π), kde α je obsah zásobníka, x je doteraz neprečítaná časť vstupného reťazca, π je doteraz vytvorená časť výstupného reťazca. Konfiguráciu $(\#, w, e)$ budeme nazývať *začiatočná konfigurácia*, konfiguráciu $(\# S, e, \pi)$ *koncová konfigurácia*.

Príklad 6.42. Urobíme syntaktickú analýzu reťazca $a + a * a$. Pri analýze využijeme rozkladovú tabuľku z príkladu 6.41.

$$
\begin{array}{llll}
(\#, a+a*a, e) & \vdash (\# E' & , a+a*a, & 3) \\
 & \vdash (\# E'T' & , a+a*a, & 36) \\
 & \vdash (\# E'T'a, & +a*a, & 36) \\
 & \vdash (\# E'T'F, & +a*a, & 368) \\
 & \vdash (\# E'T & , +a*a, & 3684) \\
 & \vdash (\# E & , +a*a, & 36841) \\
 & \vdash (\# E+ & , a*a, & 36841) \\
 & \vdash (\# E' & , a*a, & 368412) \\
 & \vdash (\# E'T' & , a*a, & 3684126) \\
 & \vdash (\# E'T'a, & *a, & 3684126) \\
 & \vdash (\# E'T'F, & *a, & 36841268) \\
 & \vdash (\# E'T & , *a, & 368412684) \\
 & \vdash (\# E'T* & , a, & 368412684) \\
 & \vdash (\# E'T' & , a, & 3684126845) \\
 & \vdash (\# E'T'a, & e, & 3684126845) \\
 & \vdash (\# E'T'F, & e, & 36841268458) \\
 & \vdash (\# E'T & , e, & 368412684584) \\
 & \vdash (\# E & , e, & 3684126845841)
\end{array}
$$

Zvláštnym prípadom silných LR(k) gramatík sú silné LR(0) gramatiky. Pre tieto gramatiky stačí na rozhodnutie o ďalšej vykonávanej operácii pri syntaktickej analýze informácia o jednom symbole na vrchu zásobníka. Silná LR(0) gramatika $G = (N, T, P, S)$ má tieto vlastnosti:

1. Pravá strana každého pravidla zodpovedajúcej rozšírenej gramatiky G' končí iným symbolom.
2. Symbol, ktorý je na konci pravej strany niektorého pravidla, sa nevyskytuje nikde inde na pravej strane žiadneho pravidla.

Z toho vyplýva, že gramatika G' neobsahuje e-pravidlá a že začiatočný symbol S' sa v G' nevyskytuje na pravej strane žiadneho pravidla. To znamená, že

symboly, ktoré sa vyskytujú na koncoch pravých strán jednotlivých pravidiel, určujú jednoznačne, kedy a pomocou ktorého pravidla sa má vykonať redukcia. Ak sa symbol X, ktorý je na konci pravej strany pravidla $A \to \alpha X$, objaví na vrchu zásobníka, znamená to, že sa má vykonať redukcia pomocou pravidla $A \to \alpha X$.

Príklad 6.43. Gramatika $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidlá

1. $S \to Aa$ 2. $A \to bB$
2. $A \to Ac$ 4. $B \to d$

je silná LR(0) gramatika. Gramatiku rozšírime o pravidlo 0. $S' \to S$ a vytvoríme rozkladovú tabuľku (*tab. 6.14*).

Tabuľka 6.14

F	e
S	prijatie
A	presun
B	redukcia (2)
a	redukcia (1)
b	presun
c	redukcia (3)
d	redukcia (4)
#	presun

Syntaktickú analýzu reťazca *bdca* vykonáme takto:

(#, *bdca*, *e*) ⊢ (# *b* , *dca*, *e*)
⊢ (# *bd*, *ca*, *e*)
⊢ (# *bB*, *ca*, 4)
⊢ (# *A* , *ca*, 42)
⊢ (# *Ac*, *a*, 42)
⊢ (# *A* , *a*, 423)
⊢ (# *Aa*, *e*, 423)
⊢ (# *S* , *e*, 4231)

6.2.2 Slabé LR gramatiky

Pri silných LR gramatikách sme pri syntaktickej analýze využívali iba informácie o jednom symbole na vrchu zásobníka a o najbližších k symboloch z doteraz neanalyzovanej časti vstupného reťazca. V prípade slabých LR gramatík už s touto metódou nevystačíme a musíme pri rozhodovaní o ďalšom postupe

analýzy využívať informácie o jej priebehu. Je teda zrejmé, že slabé LR gramatiky zahŕňujú aj silné LR gramatiky. Preto budeme ďalej prívlastok „slabé" vynechávať. Počas syntaktickej analýzy nejakého reťazca x, ktorý generuje gramatika G, metódou zdola nahor je v zásobníku umiestnený reťazec zodpovedajúci predpone nejakej pravej vetnej formy, ktorá sa vyskytuje v pravej derivácii reťazca x v gramatike G. Ak je v gramatike $G = (N, T, P, S)$ možná pravá derivácia $S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w \overset{*}{\Rightarrow} xw$, tak pravú vetnú formu $\alpha\beta w$ možno redukovať pomocou pravidla $A \to \beta$ na pravú vetnú formu $\alpha A w$. Podreťazec β je redukčné jadro vetnej formy $\alpha\beta w$, $\alpha, \beta \in (N \cup T)^*$, $w \in T^*$.

Definícia 6.19. Nech $G = (N, T, P, S)$ je bezkontextová gramatika a $G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$ je jej zodpovedajúca rozšírená gramatika. Budeme hovoriť, že G je *LR*(k), $k \geq 0$, ak z podmienok (uvedené derivácie sú pravé)

1. $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w$
2. $S' \overset{*}{\Rightarrow} \gamma B x \Rightarrow \alpha\beta y$
3. $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$

vyplýva, že $\alpha A y = \gamma B x$, t. j. $\alpha = \gamma$, $A = B$ a $x = y$.

Inými slovami to znamená, že na základe informácií o reťazci α a vopred prezeranom reťazci dĺžky k možno jednoznačne rozhodnúť o tom, že je potrebné vykonať redukciu reťazca pomocou pravidla $A \to \beta$.

Definícia 6.20. Predpokladajme, že $S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w$ je pravá derivácia v bezkontextovej gramatike $G = (N, T, P, S)$. Reťazec γ je *perspektívna predpona* v G, ak je predponou $\alpha\beta$. To znamená, že γ je reťazec, ktorý je takou predponou nejakej pravej vetnej formy, že nepresiahne pravý koniec redukčného jadra v tejto pravej vetnej forme. Ak $\gamma = \alpha\beta$, tak γ je *úplná perspektívna predpona*.

Perspektívne predpony sa objavujú počas syntaktickej analýzy metódou zdola nahor v zásobníku. V prípade, že sa v zásobníku objaví úplná perspektívna predpona, možno vykonať redukciu.

Okrem LR(k) gramatík sa budeme zaoberať nasledujúcimi podtriedami LR(k) gramatík:

a) LR(0) gramatiky,
b) jednoduché LR(k) gramatiky, (SLR(k) gramatiky),
c) LALR(k) gramatiky.

Dôvody zavedenia týchto podtried LR(k) gramatík sú nasledujúce:

— konštrukcia syntaktického analyzátora pre tieto podtriedy je jednoduchšia než pre LR(k) gramatiky,
— tabuľky pre syntaktický analyzátor majú menší rozsah,
— v niektorých prípadoch je syntaktická analýza rýchlejšia.

6.2.2.1 LR(0) gramatiky

LR(0) sú také gramatiky, pre ktoré možno vytvoriť deterministický syntaktický analyzátor pracujúci metódou zdola nahor, ktorý využíva na rozhodnutie o tom, akú operáciu vykonať, iba informácie o doterajšom priebehu analýzy. To znamená, že nepotrebuje žiadne informácie o tom, aké symboly sa vyskytujú na začiatku doteraz nespracovanej časti vstupného reťazca.

Príklad 6.44. Daná je gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

$S \to aAb$ $\quad A \to Aa \quad B \to e$
$S \to aaBba$ $\quad A \to b$

Ukážeme, že pri deterministickej syntaktickej analýze reťazcov, ktoré táto gramatika generuje, stačí informácia o histórii analýzy. Táto skutočnosť nie je na prvý pohľad zrejmá, pretože napríklad symbol b sa v pravidlách gramatiky vyskytuje v troch situáciách: tvorí pravú stranu pravidla $A \to b$ a súčasne sa vyskytuje ako podreťazec pravých strán pravidiel $S \to aAb$ a $S \to aaBba$.

Analýza reťazcov *abab* a *aaba* je v nasledujúcej tabuľke (*tab. 6.15*).

Tabuľka 6.15

vstup	v zásobníku	vykonávaná operácia
abab	*e*	presun *a*
bab	*a*	presun *b*
ab	*ab*	redukcia $A \to b$
ab	*aA*	presun *a*
b	*aAa*	redukcia $A \to Aa$
b	*aA*	presun *b*
e	*aAb*	redukcia $S \to aAb$
e	*S*	prijatie
aaba	*e*	presun *a*
aba	*a*	presun *a*
ba	*aa*	redukcia $B \to e$
ba	*aaB*	presun *b*
a	*aaBb*	presun *a*
e	*aaBba*	redukcia $S \to aaBba$
e	*S*	prijatie

Z uvedených dvoch príkladov je vidieť, že každá redukcia sa vykonáva pri určitom obsahu zásobníka.

Ďalšia tabuľka (*tab. 6.16*) ukazuje, aké reťazce budú obsahom zásobníka v okamihu redukcie pomocou určitého pravidla. Tieto reťazce sú vždy úplnými perspektívnymi predponami.

Tabuľka 6.16

pravidlo	obsah zásobníka pri redukcii
$S \to aAb$	*aAb*
$S \to aaBba$	*aaBba*
$A \to Aa$	*aAa*
$A \to b$	*ab*
$B \to e$	*aa*

Vidíme, že uvedený príklad je jednoduchý, pretože obsahom zásobníka pri redukcii (úplnou perspektívnou predponou) je jediný reťazec pre každé pravidlo. Úplných perspektívnych predpôn môže byť však pre určité pravidlo väčší počet, dokonca je možné, aby úplných perspektívnych predpôn bolo nekonečne veľa. Je však dokázané, že množiny úplných perspektívnych predpôn sú regulárne. To znamená, že na analýzu perspektívnych predpôn možno vytvoriť konečný automat. Tento automat sa nazýva *charakteristický automat syntaktického analyzátora* pre LR gramatiky, skrátene *LR automat.* Použitie LR automatu na analýzu perspektívnych predpôn zjednodušuje syntaktickú analýzu v tom zmysle, že pri rozhodovaní o tom, aká operácia sa má vykonať netreba prezerať obsah zásobníka, ale stačí zistiť, v akom stave je LR automat.

LR automat pre gramatiku G rozšírenú o pravidlo $S' \to S$ je na *obr. 6.3.*

Obr. 6.3. LR automat

Automat na *obr. 6.3* bol vytvorený tak, aby pre každú úplnú perspektívnu predponu bola v automate možná postupnosť prechodov zo začiatočného stavu do koncového stavu, ktorá zodpovedá okamihu redukcie pomocou určitého pravidla. Preto sú koncové stavy rôzne a každý je označený pravidlom, pomocou ktorého sa vykonáva redukcia.

Syntaktická analýza sa vykonáva takto: Čítame vstupný reťazec a prechádzame po zodpovedajúcej ceste. Pritom do zásobníka ukladáme stavy, ktorými

sme prešli. Keď prídeme do koncového stavu, urobíme redukciu. To znamená, že v zásobníku nahradíme stavy, ktoré zodpovedajú pravej strane redukčného pravidla, stavom zodpovedajúcim neterminálnemu symbolu na ľavej strane redukčného pravidla. V LR automate si túto operáciu môžeme predstaviť tak, že z koncového stavu sa vrátime až do stavu, v ktorom sme boli pred spracovaním prvého symbolu na pravej strane redukčného pravidla. V tomto stave musí začínať hrana, ktorá je ohodnotená neterminálnym symbolom z ľavej strany redukčného pravidla. Po tejto hrane prejdeme do ďalšieho stavu. Aby sme mohli jednoduchšie určiť okamih, kedy syntaktická analýza končí, rozširujeme gramatiku o nový začiatočný symbol S' a pravidlo $S' \to S$. Úplná perspektívna predpona pre toto pravidlo je vždy S. Redukciu pomocou tohto pravidla budeme považovať za ukončenie syntaktickej analýzy.

Stavy LR automatu môžeme označiť mnemotechnicky takto: stav, do ktorého smerujú hrany ohodnotené symbolom X označujeme X_i, a pritom index i vyberieme tak, aby označenie stavov bolo jednoznačné. Začiatočný stav budeme označovať symbolom $\#$.

Na vytvorenie LR automatu môžeme použiť tri metódy:
1. Vytvorenie súboru množín LR položiek. Funkcia GOTO predstavuje prechodový diagram LR automatu.
2. Vytvorenie LR automatu priamo na základe gramatiky.
3. Vytvorenie sústavy regulárnych rovníc, ktorých riešením získame regulárne výrazy opisujúce množiny úplných perspektívnych predpôn, a potom vytvorenie konečného automatu pre tieto regulárne výrazy.

Prvú metódu si v ďalšom rozoberieme. Druhá a tretia metóda je opísaná v [2], [9].

LR(0) položka je pravidlo gramatiky, v ktorom je označená určitá pozícia na jeho pravej strane. Na označenie pozície budeme používať symbol . (bodka). Napríklad $A \to \alpha . \beta$.

Množina LR(0) položiek obsahuje LR(0) položky, ktoré majú pred bodkou rovnaký symbol. Množina LR(0) položiek opisuje stav analýzy v okamihu, keď bol do zásobníka uložený určitý symbol. Symboly za bodkou, v množine LR(0) položiek, sú tie symboly, ktoré možno vkladať do zásobníka pri prechode z daného stavu do ďalšieho stavu. Medzi LR(0) položkami sú významné predovšetkým tieto dva druhy položiek:

— položky, ktoré majú za bodkou terminálny symbol a predstavujú situáciu, keď sa bude vykonávať presun,

— položky, ktoré majú bodku umiestnenú na konci pravej strany pravidla a predstavujú situáciu, keď sa bude robiť redukcia.

V súbore množín LR(0) položiek existuje pre každú množinu a každý jej symbol za bodkou nasledovník. Pri konštrukcii súboru množín LR(0) položiek

vytvoríme najskôr začiatočnú množinu a potom všetkých jej nasledovníkov. Rovnakú operáciu urobíme pre každú množinu LR(0) položiek.

Množinu LR(0) položiek vytvárame tak, že vždy vytvoríme jej základ a potom uzáver. Pri vytváraní uzáveru pridávame do množiny LR(0) položiek položky s bodkou na začiatku pravej strany, ktoré majú na ľavej strane neterminálny symbol vyskytujúci sa v niektorej LR(0) položke bezprostredne za bodkou. Celý postup vytvárania súboru LR(0) položiek opisuje algoritmus 6.15.

Algoritmus 6.15.

Vypočet súboru množín LR(0) položiek.
Vstup: bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: súbor $\mathscr{L}$ množín LR(0) položiek pre gramatiku G.
Metóda:
1. Vytvoríme rozšírenú gramatiku G':

   $G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$, kde $S' \notin N$
2. Začiatočnú množinu LR(0) položiek $\#$ vytvoríme takto:

   a) $\# = \{S' \to .S\}$,
   b) ak $A \to .B\alpha \in \#$, $B \in N$ a $B \to \beta \in P$, tak $\# = \# \cup \{B \to .\beta\}$,
   c) opakujeme krok b) tak dlho, pokým možno do $\#$ pridávať nové položky,
   d) $\mathscr{L} = \{\#\}$, $\#$ je začiatočná množina.
3. Ak sme vytvorili množinu LR(0) položiek M_i, vytvoríme pre každý symbol $X \in (N \cup T)$, ktorý leží v niektorej LR(0) položke v M_i za bodkou, ďalšiu množinu LR(0) položiek X_j, kde j je index, väčší ako najväčší index doteraz vytvorenej množiny LR(0) položiek X_n, takto:
   a) $X_j := \{A \to \alpha X.\beta / A \to \alpha.X\beta \in M_i\}$,
   b) ak $A \to \alpha.B\beta \in X_j$, $B \in N$, $B \to \gamma \in P$, tak $X_j := X_j \cup \{B \to .\gamma\}$,
   c) opakujeme krok b) tak dlho, pokým je možné do X_j pridávať nové položky.
   d) $\mathscr{L} = \mathscr{L} \cup \{X_j\}$, GOTO $(M_i, X) = X_j$.
4. Opakujeme krok 3 pre všetky vytvorené množiny v $\mathscr{L}$ tak dlho, pokým je možné do $\mathscr{L}$ pridávať nové množiny.

Poznámka 6.5. Kroky 2a a 3a vytvárajú základy množín LR(0) položiek, opakovaním krokov 2b a 3b sa vytvárajú uzávery množín LR(0) položiek.

Príklad 6.45. Daná je gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

1. $S \to B$
2. $B \to aBb$
3. $B \to A$
4. $A \to bA$
5. $A \to c$

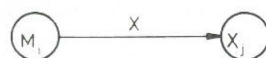
Súbor množín LR(0) položiek pre G je

$$
\begin{aligned}
\# &= \{S' \to .S \\
S &\to .B \\
B &\to .aBb \\
B &\to .A \\
A &\to .bA \\
A &\to .c\} \\
S &= \{S' \to S.\} \\
A_1 &= \{B \to A.\} \\
b_2 &= \{A \to b.A \\
A &\to .bA \\
A &\to .c\} \\
B_1 &= \{S \to B.\} \\
a &= \{B \to a.Bb \\
B &\to .aBb \\
B &\to .A \\
A &\to .bA \\
A &\to .c\} \\
c &= \{A \to c.\} \\
B_2 &= \{B \to aB.b\} \\
A_2 &= \{A \to bA.\} \\
b_1 &= \{B \to aBb.\}
\end{aligned}
$$

Teraz budeme definovať funkciu GOTO na súbore množín LR(0) položiek pre gramatiku G.

Definícia 6.21. GOTO $(M_i, X) = X_j$, ak položky tvaru $A \to \alpha.X\beta$ ležia v M_i a základ množiny X_j bol vytvorený položkami tvaru $A \to \alpha X.\beta$.

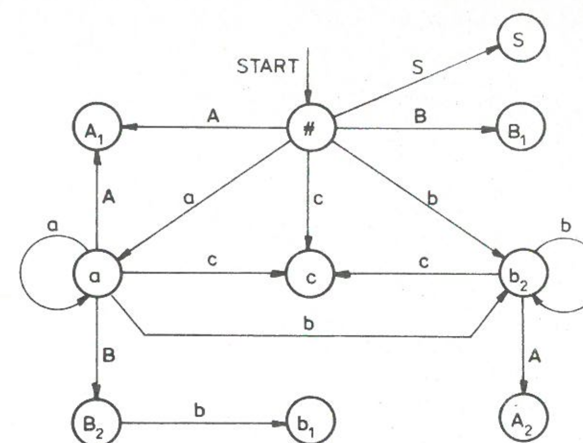
Funkciu GOTO môžeme znázorniť ako orientovaný, hranovo a vrcholovo ohodnotený graf. Funkcii GOTO $(M_i, X) = X_j$ bude zodpovedať graf na *obr. 6.4*. Funkcia GOTO je prechodová funkcia LR automatu.



Obr. 6.4. GOTO graf

Príklad 6.46. Vytvoríme graf pre funkciu GOTO na súbore množín LR(0) položiek z príkladu 6.45 (*obr. 6.5*).

Definícia 6.22. Bezkontextová gramatika $G = (N, T, P, S)$ je LR(0) gramatika, ak spĺňa túto podmienku: ak v niektorej množine súboru množín LR(0)



Obr. 6.5. GOTO graf

položiek pre gramatiku G sa vyskytuje položka $A \to \alpha.$, tak v tejto množine nie je žiadna iná položka tvaru $B \to \beta.$ alebo $B \to \beta.\gamma$ taká, že γ začína terminálnym symbolom.

Poznámka 6.6. Táto definícia je pre prípad $k = 0$ ekvivalentná s definíciou 6.19.

Teraz ukážeme, ako možno na základe súboru množín LR(0) položiek vytvoriť rozkladovú tabuľku.

Algoritmus 6.16.
Vytvorenie rozkladovej tabuľky pre LR(0) gramatiku.
Vstup: súbor $\mathscr{L}$ množín LR(0) položiek pre gramatiku $G = (N, T, P, S)$.
Výstup: rozkladová tabuľka F pre gramatiku G.
Metóda: rozkladová tabuľka F bude mať riadky označené rovnako ako množiny z $\mathscr{L}$. Pre všetky $M_i \in \mathscr{L}$ urobíme tieto kroky:

1. $F(M_i)$ = redukcia (j), ak $A \to \beta. \in M_i$ a $A \to \beta$ je j-té pravidlo v P' okrem situácie podľa 2,
2. $F(M_i)$ = prijatie, ak $S' \to S. \in M_i$,
3. $F(M_i)$ = presun v ostatných prípadoch.

Príklad 6.47. Daná je gramatika $G = (\{S', S, A, B\}, \{a, b, 0, 1\}, P, S')$, kde P obsahuje pravidlá

0. $S' \to S$	4. $A \to 0$
1. $S \to A$	5. $B \to aBbb$
2. $S \to B$	6. $B \to 1$
3. $A \to aAb$	

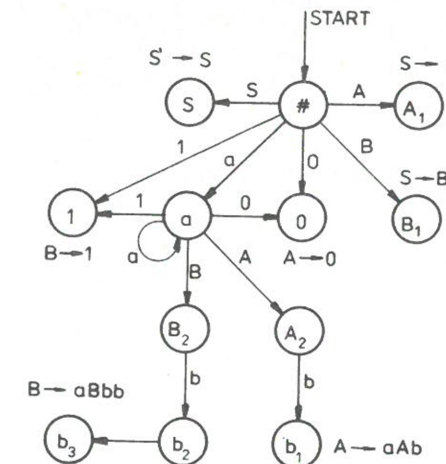
Táto gramatika nie je LL(k) pre žiadne k. Ukážeme, že je LR(0). Najskôr vytvoríme súbor množín LR(0) položiek pre G pomocou algoritmu 6.15.

$\# = \{S \to .A$
$S \to .B$
$A \to .aAb$
$A \to .0$
$B \to .aBbb$
$B \to .1$
$S' \to .S\}$
$A_1 = \{S \to A.\}$
$B_1 = \{S \to B.\}$
$a = \{A \to a.Ab$
$B \to a.Bbb$
$A \to .aAb$
$A \to .0$
$B \to .aBbb$
$B \to .1\}$
$0 = \{A \to 0.\}$
$1 = \{B \to 1.\}$
$A_2 = \{A \to aA.b\}$
$B_2 = \{B \to aB.bb\}$
$b_1 = \{A \to aAb.\}$
$b_2 = \{B \to aBb.b\}$
$b_3 = \{B \to aBbb.\}$
$S = \{S' \to S.\}$

Z tvaru množín LR(0) položiek vidíme, že gramatika G je LR(0).
Ďalej vytvoríme rozkladovú tabuľku (*tab. 6.17*) a LR automat (*obr. 6.6*).

Tabuľka 6.17

F	e
#	presun
A_1	redukcia (1)
B_1	redukcia (2)
a	presun
0	redukcia (4)
1	redukcia (6)
A_2	presun
B_2	presun
b_1	redukcia (3)
b_2	presun
b_3	redukcia (5)
S	prijatie



Obr. 6.6. LR automat

Syntaktickú analýzu pre LR(0) gramatiky budeme robiť pomocou podobného algoritmu ako pre silné LR(0) gramatiky. Rozdiel bude iba v tom, že do zásobníka nebudeme ukladať symboly gramatiky, ale ohodnotenia stavov príslušného LR automatu.

Algoritmus 6.17.
Algoritmus syntaktickej analýzy pre LR(0) gramatiky.

Vstup: rozkladová tabuľka F a LR automat pre gramatiku $G = (N, T, P, S)$, vstupý reťazec $w \in T^*$ a začiatočný symbol zásobníka # (označenie začiatočnej množiny LR(0) položiek).

Výstup: pravý rozklad v prípade, že reťazec $w \in L(G)$, inak chybová signalizácia.

Metóda: algoritmus číta symboly vstupného reťazca w, využíva zásobník a vytvára postupnosť čísel pravidiel, ktoré boli použité pri redukciách. Na začiatku je v zásobníku symbol #. Opakujeme kroky 1, 2 a 3, kým nenastane prijatie alebo chyba. X je symbol na vrchu zásobníka.

1. a) Ak $F(X)$ = presun, prečíta sa vstupný symbol a prejde sa na krok 2.
   b) Ak $F(X)$ = redukcia (i), vylúčime zo zásobníka toľko symbolov, koľko symbolov je na pravej strane i-tého pravidla (i) $A \to \alpha$ a do výstupného reťazca pridáme číslo pravidla i. Prejdeme na krok 2.
   c) Ak $F(X)$ = prijatie, ukončíme analýzu a výstupný reťazec je pravý rozklad vstupnej vety w v prípade, že vstupný reťazec je celý prečítaný, inak ukončíme analýzu chybovou signalizáciou.
2. Ak Y je symbol, ktorý sa má uložiť do zásobníka (prečítaný symbol v kroku 1. a alebo ľavá strana pravidla použitého pri redukcii v kroku 1. b a X je symbol na vrchu zásobníka, tak:

a) Ak GOTO $(X, Y) = Z$, tak na vrch zásobníka uložíme Z a prejdeme na krok 1.

b) Ak GOTO $(X, Y) =$ nedefinované, ukončíme analýzu chybovou signalizáciou.

Príklad 6.48. Urobíme analýzu reťazca $aa0bb$ pomocou rozkladovej tabuľky a LR automatu z príkladu 6.47.

$$
\begin{aligned}
(\#, aa0bb, e) &\vdash (\# a, & a0bb, & \; e)\\
&\vdash (\# aa, & 0bb, & \; e)\\
&\vdash (\# aa0, & bb, & \; e)\\
&\vdash (\# aaA_2, & bb, & \; 4)\\
&\vdash (\# aaA_2b, & b, & \; 4)\\
&\vdash (\# aA_2, & b, & \; 43)\\
&\vdash (\# aA_2b, & e, & \; 43)\\
&\vdash (\# A_2, & e, & \; 433)\\
&\vdash (\# S, & e, & \; 4331)
\end{aligned}
$$

6.2.2.2 Jednoduché LR(k) gramatiky

V predchádzajúcom článku sme sa zaoberali konštrukciou syntaktického analyzátora pre LR(0) gramatiky. Podmienku definície LR(0) gramatík spĺňa veľmi malá trieda gramatík. Dosť často sa stáva, že v niektorej množine LR(0) položiek sa vyskytuje položka tvaru $A \to \alpha.$, ktorá predstavuje redukciu, no aj iná redukčná položka tvaru $B \to \beta.$ alebo položka, ktorá predstavuje presun tvaru $C \to \gamma.a\delta$. Takúto situáciu uvedieme v príklade.

Príklad 6.49. Daná je gramatika $G = (\{E', E, T, F\}, \{+, *, (,), a\}, P, E')$, kde množina P obsahuje pravidlá

0. $E' \to E$
1. $E \to E + T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to (E)$
6. $F \to a$

Súbor množín LR(0) položiek pre gramatiku G obsahuje tieto množiny:

$$
\begin{aligned}
\# = \{&E' \to .E\\
&E \to .E + T\\
&E \to .T\\
&T \to .T * F\\
&T \to .F\\
&F \to .(E)\\
&F \to .a\}
\end{aligned}
$$

$$
\begin{aligned}
E_1 = \{&E' \to E.\\
&E \to E. + T\}\\
T_1 = \{&E \to T.\\
&T \to T. * F\}\\
F_1 = \{&T \to F.\}\\
a = \{&F \to a.\}\\
(= \{&F \to (.E)\\
&E \to .E + T\\
&E \to .T\\
&T \to .T * F\\
&T \to .F\\
&F \to .(E)\\
&F \to .a\}\\
+ = \{&E \to E + .T\\
&T \to .T * F\\
&T \to .F\\
&F \to .(E)\\
&F \to .a\}\\
* = \{&T \to T * .F\\
&F \to .(E)\\
&F \to .a\}\\
E_2 = \{&F \to (E.)\\
&E \to E. + T\}\\
T_2 = \{&E \to E + T.\\
&T \to T. * F\}\\
F_2 = \{&T \to T * F.\}\\
) = \{&F \to (E).\}
\end{aligned}
$$

Gramatika G nie je LR(0), pretože napríklad v množine E_1 sú dve položky $E' \to E.$ a $E \to E. + T$, kde nemožno rozlíšiť, či sa má vykonať prijatie (redukcia pravidlom $E' \to E$) alebo presun symbolu $+$. Takejto situácii budeme hovoriť *konflikt presun–redukcia.* Ak sa v niektorej množine LR(0) položiek vyskytnú dve rôzne položky tvaru $A \to \alpha.$ a $B \to \beta.$, hovoríme, že nastal *konflikt redukcia–redukcia.*

Konflikty v množinách LR(0) položiek môžeme odstrániť tak, že rozhodnutie o vykonávanej operácii v konfliktnom prípade urobíme na základe informácií o tom, aké symboly sa vyskytujú na začiatku doteraz nespracovanej časti vstupného reťazca. Ak ďalej uvedený postup vedie k odstráneniu konfliktov, budeme hovoriť, že daná gramatika je jednoduchá LR(k) gramatika (tiež SLR(k) gramatika), kde $k \geqq 1$ a k znamená dĺžku predpony doteraz nespracovanej časti vstupného reťazca, ktorú bolo potrebné skúmať pri odstraňovaní všetkých konfliktov v súbore množín LR(0) položiek.

Definícia 6.23. Bezkontextová gramatika $G = (N, T, P, S)$ je *jednoduchá LR(k)* (SLR(k)) gramatika, ak platí nasledujúca podmienka: nech $\mathscr{L}$ je súbor množín LR(0) položiek pre gramatiku G a $A \to \alpha.\beta$, $B \to \gamma.\delta$ sú dve rôzne LR(0) položky v ľubovoľnej množine LR(0) položiek súboru $\mathscr{L}$. Potom každé dve také položky spĺňajú aspoň jednu z uvedených podmienok:

1. Ani β, ani δ nie sú prázdne reťazce.
2. $\beta \neq e$, $\delta = e$ a $\mathrm{FOLLOW}_k(B) \cap \mathrm{FIRST}_k(\beta \mathrm{FOLLOW}_k(A)) \neq \emptyset$ pre $\beta \in T(N \cup T)^*$
3. $\beta = e$, $\delta \neq e$ a $\mathrm{FOLLOW}_k(A) \cap \mathrm{FIRST}_k(\delta \mathrm{FOLLOW}_k(B)) \neq \emptyset$ pre $\delta \in T(N \cup T)^*$.
4. $\beta = \delta = e$ a $\mathrm{FOLLOW}_k(A) \cap \mathrm{FOLLOW}_k(B) \neq \emptyset$.

Zápis $\beta \in T(N \cup T)^*$ znamená, že reťazec β začína terminálnym symbolom.

Pre SLR(k) gramatiky vytvoríme rozkladovú tabuľku F pomocou nasledujúceho algoritmu.

Algoritmus 6.18.

Konštrukcia rozkladovej tabuľky F pre SLR(k) gramatiku.

Vstup: SLR(k) gramatika $G = (N, T, P, S)$ a súbor $\mathscr{L}$ množín LR(0) položiek pre gramatiku G.

Výstup: rozkladová tabuľka F pre gramatiku G.

Metóda: rozkladová tabuľka F bude mať riadky označené rovnako ako množiny položiek z $\mathscr{L}$. Stĺpce tabuľky budú označené reťazcami symbolov $u \in T^{*k}$.

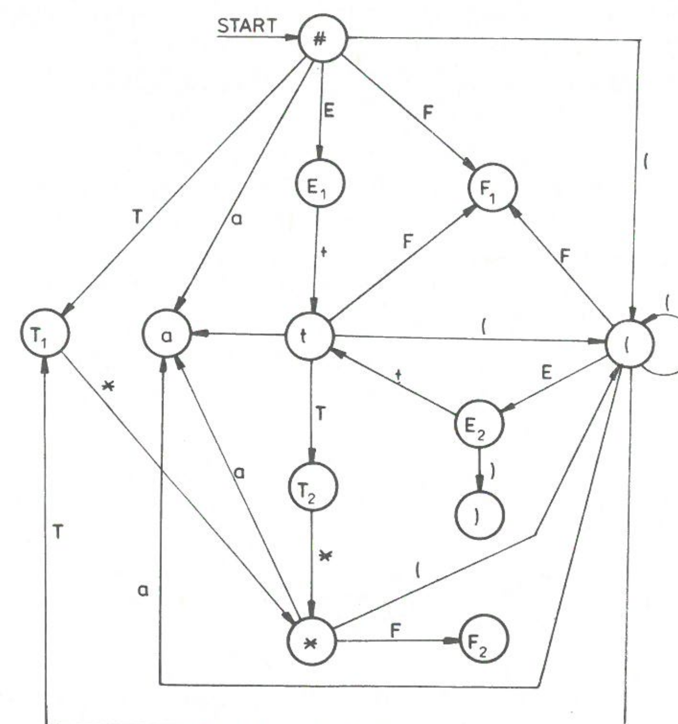
1. $F(M_i, u)$ = presun, ak $A \to \beta_1.\beta_2 \in M_i$, $\beta_2 \in T(N \cup T)^*$ a $u \in \mathrm{FIRST}_k(\beta_2 \mathrm{FOLLOW}_k(A))$.
2. $F(M_i, u)$ = redukcia (j), ak $j \geqq 1$ a $A \to \beta. \in M_i$, $A \to \beta$ je j-té pravidlo v P a $u \in \mathrm{FOLLOW}_k(A)$.
3. $F(M_i, e)$ = prijatie, ak $S' \to S. \in M_i$.
4. $F(M_i, u)$ = chyba vo všetkých ostatných prípadoch.

Teraz ukážeme na príklade, ako môžeme pomocou algoritmu 6.18 vytvoriť rozkladovú tabuľku pre gramatiku G z príkladu 6.49.

Príklad 6.50. Vytvoríme najskôr funkciu GOTO (A, X) pre $X \in N \cup T$ a pre súbor množín položiek z príkladu 6.49. Táto funkcia vyjadrená vo forme LR automatu je na *obr. 6.7*.

Pomocou algoritmu 6.18 vytvoríme rozkladovú tabuľku F pre gramatiku z príkladu 6.49. V tabuľke použijeme tieto skratky: P — presun, R_i — redukcia (i), A — prijatie (akceptovanie), prázdne položky predstavujú chybové situácie (*tab. 6.18*).

Teraz ešte musíme uviesť upravený algoritmus syntaktickej analýzy, pretože nemôžme použiť algoritmus pre LR(0) gramatiky. Dôvodom je skutočnosť, že rozkladová tabuľka pre SLR(k) gramatiky je dvojrozmerná.



Obr. 6.7. LR automat

Tabuľka 6.18

F	a	+	*	(	)	e
#	P			P		
E_1		P				A
T_1		R_2	P		R_2	R_2
F_1		R_4	R_4		R_4	R_4
a		R_6	R_6		R_6	R_6
(	p			P		
+	P			P		
*	P			P		
E_2		P			P	
T_2		R_1	P		R_1	R_1
F_2		R_3	R_3		R_3	R_3
)		R_5	R_5		R_5	R_5

Algoritmus 6.19.

Algoritmus syntaktickej analýzy pre SLR(k) gramatiky (použiteľný i pre LALR(k) a LR(k) gramatiky opísané ďalej).

Vstup: rozkladová tabuľka F a LR automat pre gramatiku $G = (N, T, P, S)$,

vstupný reťazec $w \in T^*$ a začiatočný symbol zásobníka $\#$ (označenie začiatočnej množiny LR položiek).

Výstup: pravý rozklad v prípade, že $w \in L(G)$, inak chybová signalizácia.

Metóda: algoritmus číta symboly vstupného reťazca w, využíva zásobník a vytvára postupnosť čísel pravidiel, ktoré sa použili pri redukciách. V zásobníku je na začiatku symbol $\#$. Opakujeme kroky 1, 2 a 3, pokým nenastane prijatie alebo chyba. X je symbol na vrchu zásobníka.

1. Určíme reťazec prvých k symbolov z doteraz nespracovanej časti vstupného reťazca a označíme ho u.
2. a) ak $F(X, u) =$ presun, prečíta sa vstupný symbol a prejdeme na krok 3.
   b) ak $F(X, u) =$ redukcia (i), vylúčime zo zásobníka toľko symbolov, koľko je symbolov na pravej strane i-tého pravidla (i) $A \to \alpha$ a do výstupného reťazca pridáme číslo pravidla i. Prejdeme na krok 3.
   c) ak $F(X, e) =$ prijatie, ukončíme analýzu a výstupný reťazec je pravý rozklad vstupnej vety w v prípade, že vstupný reťazec je celý prečítaný, inak ukončíme analýzu chybovou signalizáciou.
   d) ak $F(X, u) =$ chyba, ukončíme analýzu chybovou signalizáciou.
3. Ak Y je symbol, ktorý má byť uložený do zásobníka (prečítaný symbol v 2 a) alebo ľavá strana pravidla použitého pri redukcii v 2 b, a X je symbol na vrchu zásobníka, tak:
   a) ak GOTO $(X, Y) = Z$, tak uložíme Z na vrchu zásobníka a prejdeme na krok 1.
   b) ak GOTO $(X, Y) =$ chyba, ukončíme analýzu chybovou signalizáciou.

Príklad 6.51. urobíme syntaktickú analýzu reťazca $a + a{*}a$ v gramatike G z príkladu 6.49 podľa rozkladovej tabuľky a LR automatu z príkladu 6.50.

$$(\#, a + a * a, e) \vdash (\# a, + a * a, e)$$
$$\vdash (\# F_1, + a * a, 6)$$
$$\vdash (\# T_1, + a * a, 64)$$
$$\vdash (\# E_1, + a * a, 642)$$
$$\vdash (\# E_1 +, a * a, 642)$$
$$\vdash (\# E_1 + a, * a, 642)$$
$$\vdash (\# E_1 + F_1, * a, 6426)$$
$$\vdash (\# E_1 + T_2, * a, 64264)$$
$$\vdash (\# E_1 + T_2 *, a, 64264)$$
$$\vdash (\# E_1 + T_2 * a, e, 64264)$$
$$\vdash (\# E_1 + T_2 * F_2, e, 642646)$$
$$\vdash (\# E_1 + T_2, e, 6426463)$$
$$\vdash (\# E_1, e, 64264631)$$

6.2.2.3 LALR gramatiky

V predchádzajúcom pododseku sme sa zaoberali metódou konštrukcie syntaktického analyzátora pre SLR(k) gramatiky. Definícia SLR(k) gramatiky vyžadovala, aby bolo možné odstrániť všetky konflikty v súbore množínLR(0) položiek pomocou výpočtu množín FOLLOW$_k(A)$, kde A bol neterminálny symbol na ľavej strane pravidla, ktoré malo byť použité pri redukcii. Niekedy sa ale stane, že množina FOLLOW má príliš veľkú mohutnosť na to, aby bolo možné odstrániť všetky konflikty. Ukážeme takú situáciu na príklade.

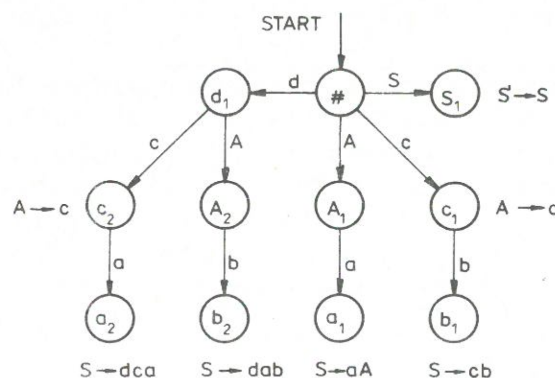
Príklad 6.52. Daná je gramatika $G = (\{S', S, A\}, \{a, b, c, d\}, P, S')$, kde P obsahuje pravidlá

0. $S' \to S$
1. $S \to Aa$
2. $S \to dAb$
3. $S \to cb$
4. $S \to dca$
5. $A \to c$

Súbor množín LR(0) položiek pre túto gramatiku má tvar:

$$\# = \{S' \to .S,\; S \to .Aa,\; S \to .dAb,\; S \to .cb,\; S \to .dca,\; A \to .c\}$$
$$S_1 = \{S' \to S.\}$$
$$A_1 = \{S \to A.a\}$$
$$d_1 = \{S \to d.Ab,\; S \to d.ca,\; A \to .c\}$$
$$c_1 = \{S \to c.b,\; A \to c.\}$$
$$a_1 = \{S \to Aa.\}$$
$$A_2 = \{S \to dA.b\}$$
$$c_2 = \{S \to dc.a,\; A \to c.\}$$
$$b_1 = \{S \to cb.\}$$
$$b_2 = \{S \to dAb.\}$$
$$a_2 = \{S \to dca.\}$$

LR automat zodpovedajúci funkcii GOTO je na *obr. 6.8*. Je vidieť, že v stavoch c_1 a c_2 sú konflikty opísaným postupom neodstrániteľné, pretože FOLLOW$(A) = \{a, b\}$ a zo stavu c_1 vychádza hrana ohodnotená symbolom b, zo stavu c_2 vychádza hrana ohodnotená symbolom a.



Obr. 6.8. LR automat s konfliktmi v stavoch *c*1 a *c*2

Konflikty v stavoch c_1 a c_2 môžeme odstrániť takto:

a) V stave c_1:
Pre vstupný symbol b urobíme presun. Redukciu pomocou pravidla $A \to c$ urobíme vtedy, ak vo vstupnom reťazci bude symbol a. Dôvod spočíva v tom, že pri redukcii pomocou pravidla $A \to c$ v stave c_1 sa vykoná návrat do stavu #, a potom prechod do stavu A_1. Z tohto stavu je možný prechod iba pri presune symbolu a.

b) V stave c_2:
Pre vstupný symbol a urobíme presun. Redukciu pomocou pravidla $A \to c$ urobíme vtedy, ak vo vstupnom reťazci bude symbol b. Dôvod znova spočíva v tom, že po redukcii pomocou pravidla $A \to c$ prejdeme do stavu A_2, z ktorého je možný prechod iba pri presune symbolu b.

Z uvedeného príkladu je zrejmé, že rozhodnutie o tom, či urobiť v stave c_1 alebo c_2 redukciu alebo presun, nemôžeme urobiť použitím množiny FOLLOW. Pre stav c_1 sme určili, že redukcia sa bude robiť pre symbol a, pre stav c_2 je to symbol b. Oba tieto symboly sú prvky množiny FOLLOW(A). Symboly a, b v tomto prípade tvoria množiny reťazcov, ktoré budeme nazývať množiny vopred prezeraných reťazcov a označovať LA (look a head). Teda $LA_1 \to (c_1, A) = \{a\}$ a $LA_1(c_1, A) = \{b\}$. Pritom obe tieto množiny sú podmnožinami množiny FOLLOW(A). Môžeme teda sformulovať postup odstránenia konfliktov v charakteristickom automate pre prípad, že gramatika nie je jednoduchá LR gramatika:

a) Konflikt redukcia-redukcia pre pravidlá $A \to \alpha$ a $B \to \beta$ v stave q odstránime tak, že určíme množiny $LA_1(q, A)$ a $LA_1(q, B)$. V prípade, že budú mať prázdny prienik, tak redukciu pomocou pravidla $A \to \alpha$ urobíme vtedy, ak na vstupe bude reťazec z $LA_1(q, A)$, a redukciu pomocou pravidla $B \to \beta$ urobíme v prípade, že na vstupe je reťazec z $LA_1(q, B)$.

b) Konflikt redukcia-presun v stave q pre pravidlo $A \to \alpha$ a symbol b odstránime tak, že určíme $LA_1(q, A)$. V prípade, že b nie je v $LA_1(q, A)$, urobíme redukciu pomocou pravidla $A \to \alpha$ vtedy, ak na vstupe bude reťazec z $LA_1(q, A)$. Ak na vstupe bude symbol b, urobíme presun.

Ak tento postup pre danú gramatiku G vedie k cieľu, gramatika G je LALR(1) gramatika.

Množina vopred prezeraných reťazcov v gramatike G označená $LA_k(q, A)$, kde q je stav charakteristického automatu a A je neterminálny symbol z N, obsahuje reťazce terminálnych symbolov, ktoré sa môžu vyskytnúť na začiatku doteraz nespracovanej časti vstupného reťazca v prípade, že automat je v stave q, v ktorom má byť urobená redukcia pomocou pravidla $A \to \alpha$. Množina $LA_k(q, A)$ môže obsahovať aj prázdny reťazec v prípade, ak je automat v stave q a v zásobníku je reťazec zodpovedajúci celej pravej vetnej forme.

Definícia 6.24. $LA_k(q, A) = \{x / S \overset{*}{\Rightarrow} \beta Axw \Rightarrow \beta\alpha xw,\ A \to \alpha \in P,\ x \in T^*,\ |x| = k,$ $\beta \in (N \cup T)^*,\ w \in T^*$, q je stav, v ktorom sa nachádza charakteristický automat po spracovaní reťazca $\beta\alpha\} \cup \{x / S \overset{*}{\Rightarrow} \beta Ax \Rightarrow \beta\alpha x,\ |x| < k$, ostatné podmienky sú rovnaké$\}$.

Ďalej ukážeme, že na základe LR automatu možno vytvoriť množiny vopred prezeraných reťazcov.

Algoritmus 6.20.
Výpočet množín LA_1.
Vstup: LR automat pre gramatiku $G = (N, T, P, S)$, stav q, neterminálny symbol A.
Výstup: množina $LA_1(q, A)$.
Metóda: stav q musí byť ohodnotený pravidlom $A \to \alpha$. Budeme predpokladať, že toto pravidlo so symbolom A na ľavej strane je iba jediné, pretože po ohodnotení ďalším pravidlom tvaru $A \to \beta$ vznikne neodstrániteľný konflikt.

1. Položíme začiatočný stav $p = q$, $LA_1(q, A) = \emptyset$, $Q = \emptyset$.
2. Zo stavu p budeme postupovať proti smeru prechodov v LR automate po všetkých cestách zodpovedajúcich obrátenej pravej strane pravidla $A \to \alpha$. Potom urobíme prechod po hrane ohodnotenej symbolom A. Tak sa môžeme dostať do jedného z množiny stavov $R = \{r_1, r_2, \ldots, r_n\}$. Položíme $Q = Q \cup R$.
3. Pre každý stav z Q urobíme práve raz tieto operácie:
   a) Ak zo stavu r_i vychádzajú hrany ohodnotené terminálnymi symbolmi, pridáme ich do $LA_1(q, A)$.
   b) Ak stav r_i je ohodnotený pravidlom $S' \to S$, vložíme do $LA_1(q, A)$ prázdny reťazec.
   c) Ak stav r_i je ohodnotený e-pravidlom tvaru $A \to e$, pridáme do Q stav, do ktorého vedie zo stavu r_i hrana ohodnotená symbolom A.

d) Ak stav r_i je ohodnotený iným pravidlom ako $S' \to S$, vrátime sa na krok 2 pre $p = r_i$ a pre ľavé strany všetkých pravidiel, ktorými je ohodnotený stav r_i.

Teraz možno definovať LALR(k) gramatiky.

Definícia 6.25. Bezkontextová gramatika $G = (N, T, P, S)$ je LALR(k) gramatika, ak platí: Nech $\mathscr{L}$ je súbor množín LR(0) položiek pre gramatiku G. Nech $A \to \alpha.\beta$ a $B \to \gamma.\delta$ sú rôzne LR(0) položky v ľubovoľnej množine M LR(0) položiek súboru $\mathscr{L}$. Potom každé dve také položky z množiny M musia spĺňať aspoň jednu z nasledujúcich podmienok:

1. Ani β, ani δ nie sú prázdne reťazce.
2. $\beta \neq e$, $\delta = e$ a $LA_k(M, B) \cap \mathrm{FIRST}_k(\beta LA_k(M, A)) = \emptyset$ pre $\beta \in T(N \cup T)^*$
3. $\beta = e$, $\delta \neq e$ a $LA_k(M, A) \cap \mathrm{FIRST}_k(\delta LA_k(M, B)) = \emptyset$ pre $\delta \in T(N \cup T)^*$.
4. $\beta = \delta = e$ a $LA_k(M, A) \cap LA_k(M, B) = \emptyset$.

Pre LALR(k) gramatiky vytvoríme rozkladovú tabuľku pomocou nasledujúceho algoritmu.

Algoritmus 6.21.

Vytvorenie rozkladovej tabuľky pre LALR(k) gramatiku.

Vstup: súbor $\mathscr{L}$ množín LR(k) položiek pre gramatiku G = (N, T, P, S).

Výstup: rozkladová tabuľka F pre gramatiku G.

Metóda: rozkladová tabuľka F bude mať riadky označené rovnako ako množiny z $\mathscr{L}$. Stĺpce budú označeno reťazcami symbolov $u \in T^*$. Pre všetky $M_i \in \mathscr{L}$ urobíme kroky:

1. $F(M_i, u)$ = presun ak $A \to \beta_1.\beta_2 \in M_i$, $\beta_2 \in T(N \cup T)^*$ a $u \in \mathrm{FIRST}_k(\beta_2 LA_k(M_i, A))$,
2. $F(M_i, u)$ = redukcia (j), ak $A \to \beta. \in M_i$ a $A \to \beta$ je j-té pravidlo v P' (okrem situácie podľa bodu 3).
3. $F(M_i, e)$ = prijatie, ak $S' \to S. \in M_i$
4. $F(M_i, u)$ = chyba v ostatných prípadoch.

Príklad 6.53. Pre gramatiku z príkladu 6.52 vytvoríme rozkladovú tabuľku. Súbor množín LR(0) položiek pre túto gramatiku sme vytvorili v príklade 6.52. Konflikty v stavoch c_1, a c_2 môžeme odstrániť, pretože platí

$$LA_1(c_1, A) = a$$
$$LA_1(c_2, A) = b$$

Preto v stavoch c_1 a c_2 môžeme urobiť rozhodnutie o vykonánej operácii. Rozkladová tabuľka *tab. 6.19.* má tvar:

Tabuľka 6.19

F	a	b	c	d	e
#			P	P	
S_1					A
A_1	P				
d_1			P		
c_1	R_5	P			
a_1					R_1
A_2			P		
c_2	P	R_5			
b_1					R_3
b_2					R_2
a_2					R_4

Ak porovnáme rozkladové tabuľky a LR automaty pre SLR(k) gramatiky a pre LALR(k) gramatiky, zistíme, že ich rozsah je rovnaký. Rozdiel medzi SLR(k) gramatikami a LALR(k) gramatikami je iba v zložitosti výpočtu množín FOLLOW a LA.

6.2.2.4 LR(k) gramatiky

Pri konštrukcii syntaktického analyzátora pre gramtiku, ktorá je LR(k), ale pritom nie je LALR(k), nevystačíme s konštrukciou LR automatu doterajším spôsobom a prípadným odstraňovaním konfliktov, ako to bolo pri SLR(k) a LALR(k) gramatikách.

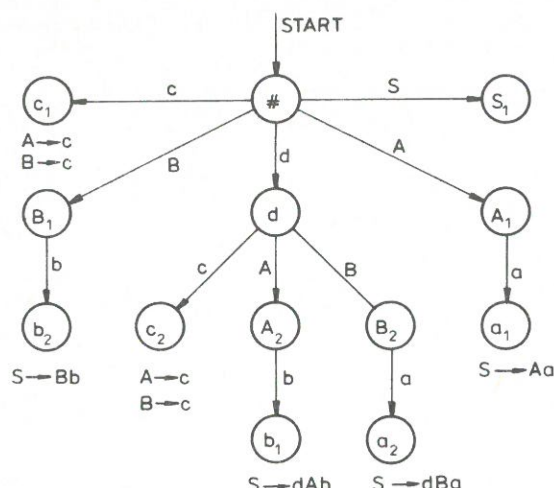
Tu uvedieme postup konštrukcie syntaktického analyzátora pre všeobecné LR gramatiky.

Príklad 6.54. Daná je gramatika $G = (\{S', S, A, B\}, \{a, b, c, d\}, P, S')$, kde P obsahuje pravidlá

$S' \to S$	$S \to dBa$
$S \to Aa$	$A \to c$
$S \to dAb$	$B \to c$
$S \to Bb$	

LR automat pre túto gramatiku je na *obr. 6.9*.

Keby sme vytvorili LR automat už opísaným spôsobom, dostali by sme automat, ktorý by mal oproti automatu na *obr. 6.9* stavy c_1 a c_2 nahradené jediným stavom, a vznikol by neodstrániteľný konflikt. Uvedieme preto metódu konštrukcie LR automatu pre všeobecnú LR(k) gramatiku, ktorej základom je konštrukcia súboru množín LR(k) položiek.



Obr. 6.9. LR automat pre LR(1) gramatiku

Definícia 6.26. LR(k) položka v gramatike $G = (N, T, P, S)$ je objekt tvaru $[A \to \alpha.\beta, u]$, kde $A \to \alpha\beta$ je pravidlo v P, u je reťazec z FOLLOW(A). V prípade, že k sa bude rovnať nule, pôjde o LR(0) položku.

Algoritmus 6.22.
Výpočet súboru množín LR(k) položiek.
Vstup: bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: súbor $\mathscr{L}$ množín LR(k) položiek pre gramatiku G.
Metóda:

1. Vytvoríme rozšírenú gramatiku $G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$, kde $S' \notin N$,
2. Začiatočnú množinu LR(k) položiek # vytvoríme takto:
   a) $\# = \{[S' \to .S, e]\}$,
   b) ak $[A \to .B\alpha, u] \in \#$, $B \in N$ a $B \to \beta \in P$ tak $\# = \# \cup \{[B \to .\beta, x]\}$ pre všetky $x \in \text{FIRST}_k(\alpha u)$
   c) opakujeme krok b) tak dlho, pokým možno do # pridávať nové položky,
   d) $\mathscr{L} = \{\#\}$, # je začiatočná množina.
3. Ak sme skonštruovali množinu LR(k) položiek M_i, tak pre každý symbol $X \in (N \cup T)$, ktorý leží v niektorej LR(k) položke v M_i za bodkou, vytvoríme ďalšiu množinu LR(k) položiek X_j, kde j je index, väčší ako najväčší index doteraz vytvorenej množiny LR(k) položiek X_n, takto
   a) $X_j = \{[A \to \alpha X.\beta, u] / [A \to \alpha.X\beta, u] \in M_i\}$,
   b) ak $[A \to \alpha.B\beta, u] \in X_j$, $B \in N$, $B \to \gamma \in P$, tak $X_j = X_j \cup \{[B \to .\gamma, x] /$ pre všetky $x \in \text{FIRST}_k(\beta u)\}$,

   c) opakujeme krok b) tak dlho, pokým možno do X_j pridávať nové položky,
   d) $\mathscr{L} = \mathscr{L} \cup \{X_j\}$, GOTO $(M_i, X) = X_j$.
4. Opakujeme krok 3 pre všetky vytvorené množiny v $\mathscr{L}$ tak dlho, pokým možno do $\mathscr{L}$ pridávať nové množiny.

Veta 6.14. Bezkontextová gramatika $G = (N, T, P, S)$ je LR(k) gramatika pre $k \geqq 0$, ak v súbore množín LR(k) položiek nie sú žiadne konflikty.

Príklad 6.55. Pre gramatiku z príkladu 6.54 vytvoríme súbor množín LR(1) položiek.

$\# = \{[S' \to .S\ ,\ e]$
$[S \to .Aa\ ,\ e]$
$[S \to .dAb,\ e]$
$[S \to .Bb\ ,\ e]$
$[S \to .dBa,\ e]$
$[A \to .c\ ,\ a]$
$[B \to .c\ ,\ b]\}$
$S_1 = \{[S' \to S.\ ,\ e]\}$
$A_1 = \{[S \to A.a\ ,\ e]\}$
$d = \{[S \to d.Ab,\ e]$
$[A \to .c\ ,\ b]$
$[S \to d.Ba,\ e]$
$[B \to .c\ ,\ a]\}$
$B_1 = \{[S \to B.b\ ,\ e]\}$
$c_1 = \{[A \to c.\ ,\ a]$
$[B \to c.\ ,\ b]\}$
$a_1 = \{[S \to Aa.\ ,\ e]\}$
$A_2 = \{[S \to dA.b,\ e]\}$
$b_2 = \{[S \to Bb.\ ,\ e]\}$
$B_2 = \{[S \to dB.a,\ e]\}$
$b_1 = \{[S \to dAb.,\ e]\}$
$a_2 = \{[S \to dBa.,\ e]\}$
$c_2 = \{[A \to c.\ ,\ b]$
$[B \to c.\ ,\ a]\}$

V tomto súbore množín LR(k) položiek nie je žiadny konflikt, a preto daná gramatika je LR(1).

Príklad 6.56. Daná je gramatika $G = (\{S', S, A, B, C, D, E\}, \{a, b\}, P, S')$, kde P obsahuje pravidlá

$S' \to S$	$B \to aE$
$S \to AB$	$C \to ab$
$A \to a$	$D \to bb$
$B \to CD$	$E \to bba$

Ukážeme, že táto gramatika je LR(2) gramatikou. Vytvoríme súbor množín LR(2) položiek:

$$
\begin{array}{rl}
\# = & \{[S' \to .S \; , e] \\
 & [S \to .AB, e] \\
 & [A \to .a \; , ab]\} \\
S = & \{[S' \to S. \; , e]\} \\
A = & \{[S \to A.B, e] \\
 & [B \to .CD, e] \\
 & [B \to .aE \; , e] \\
 & [C \to .ab \; , bb]\} \\
a_1 = & \{[A \to a. \; , ab]\} \\
B = & \{[S \to AB., e]\} \\
C = & \{[B \to C.D, e] \\
 & [D \to .bb \; , e]\} \\
a = & \{[B \to a.E \; , e] \\
 & [E \to .bba, e] \\
 & [C \to a.b \; , e]\} \\
D = & \{[B \to CD., e]\} \\
b_1 = & \{[D \to b.b \; , e]\} \\
E = & \{[B \to aE. \; , e]\} \\
b_2 = & \{[E \to b.ba, e] \\
 & [C \to ab. \; , bb]\} \\
b_3 = & \{[D \to bb. \; , e]\} \\
b = & \{[E \to bb.a, e]\} \\
a_2 = & \{[E \to bba., e]\}
\end{array}
$$

V tomto súbore množín LR(2) položiek nie je žiadny konflikt. To znamená, že G je LR(2). Keby sme vytvorili súbor množín LR(1) položiek, vznikol by konflikt v množine b_2.

Rozkladovú tabuľku pre LR gramatiku vytvoríme pomocou algoritmu, ktorý sa podobá algoritmu na vytvorenie rozkladovej tabuľky pre LALR gramatiky. Rozdiel je iba v tom, že vopred prezerané reťazce sú priamo súčasťou LR(k) položiek

Algoritmus 6.23.
Vytvorenie rozkladovej tabuľky pre LR(k) gramatiku.
Vstup: Súbor množín LR(k) položiek $\mathscr{L}$ pre gramatiku $G = (N, T, P, S)$.
Výstup: rozkladová tabuľka F pre gramatiku G.
Metóda: rozkladová tabuľka F bude mať riadky označené rovnako ako množiny z $\mathscr{L}$. Stĺpce budú označené reťazcami symbolov $u \in T^{*k}$. Pre všetky $M_i \in \mathscr{L}$ prevedieme tieto kroky:

1. $F(M_i, u)$ = presun ak $[A \to \beta_1.\beta_2, v] \in M_i$, $\beta_2 \in T(N \cup T)^*$ a $u \in \mathrm{FIRST}_k(\beta_2 v)$.
2. $F(M_i, u)$ = redukcia (j), ak $[A \to \beta., u] \in M_i$ a $A \to \beta$ je j-té pravidlo v P' (okrem situácie podľa bodu 3).
3. $F(M_i, e)$ = prijatie, ak $[S' \to S., e] \in M_i$, a vstupný reťazec je celý prečítaný.
4. $F(M_i, u)$ = chyba v ostatných prípadoch.

Príklad 6.57. Pomocou algoritmu 6.23 vytvoríme rozkladovú tabuľku pre LR(1) gramatiku z príkladu 6.54. V príklade 6.55 je uvedený súbor LR(1) položiek pre túto gramatiku. Rozkladová tabuľka *tab. 6.20* má tento tvar.

Tabuľka 6.20

F	a	b	c	d	e
#			P	P	
S_1					A
A_1	P				
d			P		
B_1		P			
c_1	R_5	R_6			
a_1					R_1
A_2		P			
b_2					R_3
B_2	P				
b_1					R_2
a_2					R_4
c_2	R_6	R_5			

Algoritmus syntaktickej analýzy pre LR(k) gramatiky je rovnaký ako pre SLR(k) a LALR(k) gramatiky.

6.2.3 Vlastnosti LR gramatík a jazykov

V predchádzajúcich pododsekoch sme sa zaoberali metódami konštrukcie rozkladových tabuliek a algoritmami syntaktickej analýzy pre jednotlivé triedy LR(k) gramatík. V tomto odseku sa zmienime o niektorých teoretických problémoch, ktoré majú dopad na praktickú použiteľnosť teórie LR(k) gramatík pri konštrukcii syntaktických analyzátorov bezkontextových jazykov.

Definícia 6.27. Bezkontextový jazyk L sa nazýva *LR(k) jazyk*, ak existuje LR(k) gramatika G taká, že $L = L(G)$. Bezkontextový jazyk sa nazýva *LR jazyk*, ak existuje LR(k) gramatika G pre nejaké $k \geqq 0$ taká, že $L = L(G)$.

Veta 6.15. Každá LR(k) gramatika je jednoznačná.

Jednoznačnosť LR(k) gramatiky vyplýva z definície 6.19. Ak je daná pravá

derivácia $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w$, tak v gramatike G môže byť najviac jedno pravidlo $A \to \beta$ také, že vetnú formu $\alpha\beta w$ možno redukovať na $\alpha A w$. To znamená, že pre každú vetu jazyka existuje v LR(k) gramatike jediná pravá derivácia.

Veta 6.16. Pre danú bezkontextovú gramatiku a dané pevné $k \geqq 0$ je rozhodnuteľné, či G je alebo nie je LR(k) (silná LR(k), SLR(k), LALR(k)) gramatika.

Pre LR(k) gramatiky stačí vytvoriť súbor LR(k) položiek a preveriť, či v tomto súbore nie je konflikt. Podobne možno postupovať pri SLR(k) a LALR(k) gramatikách. Pre silné LR(k) gramatiky stačí urobiť kontrolu pravidiel gramatiky podľa definície 6.18.

Veta 6.17. Pre danú bekontextovú gramatiku G je nerozhodnuteľné, či je LR(k) gramatikou pre nejaké $k \geqq 0$.

Veta 6.18. Každá LL(k) gramatika je LR(k) gramatika.

Veta 6.19. Množina LR(0) gramatík je vlastnou podmnožinou množiny SLR(1) gramatík. Množina SLR(k) gramatík je vlastnou podmnožinou množiny LALR(k) gramatík. Množina LALR(k) gramatík je vlastnou podmnožinou množiny LR(k) gramatík.

Uvedené vlastnosti vyplývajú z príkladov 6.49, 6.52 a 6.54.

Veta 6.20. Existujú gramatiky, ktoré sú LR(1), ale nie sú SLR(k) pre žiadne k.

Príklad 6.58. Daná je gramatika $G = (\{Z, S, A, B, C, D\}, \{a, b, c\}, P, Z)$, kde P obsahuje pravidlá

$Z \to Sc$	$B \to C$
$S \to CbBA$	$B \to Db$
$A \to ab$	$C \to a$
$A \to Aab$	$D \to a$

Táto gramatika je LR(1). Je zrejmé, že v súbore LR(0) položiek sa vyskytnú v jednej množine položky $C \to a.$ a $D \to a.$, a pritom platí

$$\mathrm{FOLLOW}_k(C) = \mathrm{FOLLOW}_k(B) \cup \mathrm{FIRST}_k(b\mathrm{FIRST}_k(BA)c) = \\ = \mathrm{FIRST}_k((ab)^n c \cup b(ab)^n c \cup ba(ab)^n c)$$
$$\mathrm{FOLLOW}_k(D) = \mathrm{FIRST}_k(b\mathrm{FOLLOW}_k(B)) = \mathrm{FIRST}_k(b(ab)^n c)$$
pre $n \geqq 1$. Je zrejme, že
$$\mathrm{FOLLOW}_k(C) = \mathrm{FOLLOW}_k(D) = \mathrm{FIRST}_k(b(ab)^n c) \neq \emptyset$$

Preto gramatika G nie je SLR(k) pre žiadne k.

Veta 6.21. Existujú gramatiky, ktoré sú LL(1), ale nie sú SLR(1).

Príklad 6.59. Daná je gramatika $G = (\{Z, S, A, B\}, \{a, b, c\}, P, Z)$, kde P obsahuje pravidlá

$Z \to Sc$	$A \to e$
$S \to AaAb$	$B \to e$
$S \to BbBa$	

Táto gramatika je LL(1). Pretože sa však v súbore množín LR(0) položiek vyskytnú v jednej množine položky $A \to .$ a $B \to .$ a platí $\mathrm{FOLLOW}(A) \cap \cap \mathrm{FOLLOW}(B) = \{a, b\}$, nie je táto gramatika SLR(1).

Teraz sa pokúsime charakterizovať jednotlivé triedy LR jazykov. LR(0) jazyky majú bezprefixovú vlastnosť.

Definícia 6.28. Jazyk $L \subset T^*$ má *bezprefixovú vlastnosť*, keď z $x \in L$ a $xy \in L$ vyplýva, že $y = e$.

Príklad 6.60. Z definície 6. 28 vyplýva, že napríklad jazyky

$$L_1 = \{a^i / i \geqq 1\}$$
$$L_2 = \{acb^i / i \geqq 1\}$$

nemožno generovať LR(0) gramatikami, pretože nemajú bezprefixovú vlastnosť.

Množiny LR(k) jazykov pre $k \geqq 1$ sú identické, pretože platí nasledujúca veta.

Veta 6.22. Ak gramatika $G = (N, T, P, S)$ je LR(k) gramatika pre nejaké $k \geqq 1$, tak ju možno transformovať na LR(1) gramatiku. Ak jazyk $L(G)$ má bezprefixovú vlastnosť, je možné gramatiku G transformovať na LR(0) gramatiku.

Postupy, ktoré možno použiť pre transformáciu LR(k) gramatiky na LR(1) alebo LR(0) gramatiku, sú známe operácie extrakcie pravého kontextu, pohltenie terminálu a pohltenie reťazca.

Teraz ukážeme gramatiku, ktorá nie je LR(k) pre žiadne k.

Príklad 6.61. Daná je gramatika $G = (\{S', S, A, B\}, \{a, b, c\}, P, S')$, kde P obsahuje pravidlá

$S' \to S$	$A \to Aa$	$A \to Ba$
$S \to Ab$	$A \to e$	$B \to e$
$S \to Bc$		

Vytvoríme začiatočnú množinu LR(k) položiek #.

$$\# = \{[S' \to .S\ ,\ e]$$
$$[S \to .AB,\ e]$$
$$[S \to .Bc\ ,\ e]$$
$$[A \to .Aa,\ b \mid ab \mid a^2b \mid \ldots \mid a^{k-1}b \mid a^k]$$
$$[A \to .\ \ ,\ b \mid ab \mid a^2b \mid \ldots \mid a^{k-1}b \mid a^k]$$
$$[B \to .Ba,\ c \mid ac \mid a^2c \mid \ldots \mid a^{k-1}c \mid a^k]$$
$$[B \to .\ \ ,\ c \mid ac \mid a^2c \mid \ldots \mid a^{k-1}c \mid a^k]\}$$

Je vidieť, že v tejto množine LR(k) položiek je konflikt redukcia-redukcia medzi položkami $[A \to ., a^k]$ a $[B \to ., a^k]$. Pretože k je ľubovoľné, nie je gramatika G LR(k) pre žiadne k. Pritom $L(G) = \{a^n b / n \geqq 0\} \cup \{a^n c / n \geqq 0\}$ a na generovanie tohto jazyka je možné vytvoriť napr. gramatiku s pravidlami

$S' \to S$
$S \to Ab$
$S \to Ac$

$A \to Aa$
$A \to e$

ktorá je dokonca LR(0).

Cvičenia

1. Pre nasledujúce LR(0) gramatiky vytvorte LR automaty a rozkladové tabuľky.
   a) $G = (\{S, A\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

   $S \to aAa \mid bAb$
   $A \to a$

   b) $G = (\{S\}, \{a, b, c\}, P, S)$, P obsahuje pravidlá

   $S \to aSb \mid aSc \mid ab$

   c) $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

   $S \to cA \mid ccB$
   $A \to cA \mid a$
   $B \to ccB \mid b$

   d) $G = (\{S\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

   $S \to aSSb \mid aSS \mid c$

2. Najdite LR(0) gramatiky pre nasledujúce jazyky a pre nájdené gramatiky vytvorte LR automaty a rozkladové tabuľky:
   a) $L_1 = \{1^n a 0^n / n \geqq 0\} \cup \{1^n b 0^{2n} / n \geqq 0\}$,
   b) $L_2 = \{a 1^n 0^n / n > 0\} \cup \{b 1^n 0^{2n} / n \geqq 0\}$,
   c) $L_3 = \{1^n 0^m / 0 < n < m\}$,
   d) $L_4 = \{w 2 w^R / w \in \{0, 1\}^*\}$.
3. Pre nasledujúcu gramatiku vytvorte LR automat a rozkladovú tabuľku.
   $G = (\{S, A\}, \{a, +, (,)\}, P, S)$, kde P obsahuje pravidlá

   $S \to S + A \mid A$
   $A \to (S) \mid a(S) \mid a$
4. Pre nasledujúce SLR(1) gramatiky vytvorte LR automaty a rozkladové tabuľky.
   a) $G = (\{S, A, B\}, \{a, b, c, d, x\}, P, S)$, kde P obsahuje pravidlá

   $S \to bASb \mid bA$
   $A \to dSca \mid x$
   $B \to cAa \mid c$

   b) $G = (\{S, A, B, C\}, \{u, v, y, w\}, P, S)$, kde P obsahuje pravidlá

   $S \to Bv \mid vc$
   $A \to u \mid vBS$
   $B \to u \mid yw$
   $C \to Bv \mid yAw$

   c) $G = (\{S, A, B\}, \{a, b, 0, 1\}, P, S)$, kde P obsahuje pravidlá

   $S \to aA \mid bB$
   $A \to 1A0 \mid e$
   $B \to 1B00 \mid e$

5. Pre nasledujúce gramatiky vytvorte LR automaty a rozkladové tabuľky.
   a) $G = (\{S, A\}, \{a, b, (,)\}, P, S)$, kde P osbsahuje pravidlá

   $S \to (AS) \mid (b)$
   $A \to (SaA) \mid (a)$

   b) $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidlá

   $S \to aAb \mid c$
   $A \to bS \mid Bb$
   $B \to aA \mid c$

   c) $G = (\{S, A\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidlá

   $S \to AAd \mid cAd \mid b \mid ASc$
   $A \to ASc \mid cd \mid a \mid Sb$

   d) $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

   $S \to dAa \mid e$
   $A \to aSB \mid dSc$
   $B \to b \mid e$

   e) $G = (\{S, R\}, \{(, ;,), a\}, P, S)$, kde P obsahuje pravidlá

   $S \to a \mid (SR$
   $R \to ;SR \mid)$

   f) $G = (\{S, A\}, \{a, b, c, d, x\}, P, S)$, kde P obsahuje pravidlá

   $S \to aAd \mid bAc \mid axc \mid bxd$
   $A \to x$

   g) $G = (\{S, C\}, \{c, d\}, P, S)$, kde P obsahuje pravidlá

   $S \to cC$
   $C \to cC \mid d$

h) $G = (\{E, T, F\}, \{a, b, +, *, (,), e\}, P, S)$, kde P obsahuje pravidlá

$E \rightarrow E + T \mid T$
$T \rightarrow TF$
$F \rightarrow F^* \mid (E) \mid a \mid b \mid e$

i) $G = (\{S\}, \{(,)\}, P, S)$, kde P obsahuje pravidlá

$S \rightarrow S(S) \mid e$

j) $G = (\{S, A\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidlá

$S \rightarrow Aa \mid dAb \mid cb \mid dca$
$A \rightarrow c$

k) $G = (\{S\}, \{a, b\}, P, S)$, kde P obsahuje pravidlá

$S \rightarrow SaSa \mid e$

LITERATÚRA

1. AHO, A. V.—ULLMAN, J. D.: The theory of parsing, translation and compiling. Vol. 1. Parsing, Vol. 2. Compiling. New York, Prentice-Hall 1971, 1972.
2. BACKHOUSE, R. C.: Syntax of programming languages: Theory and practice. New Jersey, Prentice-Hall 1979.
3. ČEŠKA, M.—RÁBOVÁ, Z.: Gramatiky a jazyky. Učební texty VUT Brno, Praha, SNTL 1983.
4. FOSTER, J. M.: Automatická syntaktická analýza. Bratislava, Alfa 1973.
5. GLADKIJ, A. V.: Formalnyje gramatiki i jazyki. Moskva, Mir 1973.
6. HOPCROFT, J. E.—ULLMAN, J. D.: Formálne jazyky a automaty. Bratislava, Alfa 1978.
7. CHYTIL, M.: Automaty a gramatiky. Praha, SNTL 1984.
8. MELICHAR, B.: Gramatiky a jazyky. Praha, Ediční středisko ČVUT 1972.
9. MELICHAR B.: Gramatiky a jazyky — cvičení. Praha, Ediční středisko ČVUT 1983.
10. MOLNÁR, Ľ.: Gramatiky a jazyky. Bratislava, Edičné stredisko SVŠT 1983.
11. NIJHOLT, A.: Context-free grammars: Covers, normal forms and parsing. Lect. Notes Comput. Sci. Springer Verlag., Berlin 1980.
12. Překladače programovacích jazyků. Sborníky referátu, ČVTS FEL ČVUT, Praha 1974, 1978, 1982.

Register

EDÍCIA VÝPOČTOVEJ TECHNIKY

Vysokoškolská učebnica je určená predovšetkým poslucháčom elektrotechnických fakúlt vysokých škôl technických, ale poslúži aj ostatným záujemcom

DOC. RNDr. ĽUDOVÍT MOLNÁR, CSc.
DOC. RNDr. MILAN ČEŠKA, CSc.
DOC. ING. BOŘIVOJ MELICHAR, CSc.

GRAMATIKY A JAZYKY

MDT 519.682 (075.8)
681.3.06 (075.8), 800.92 (075.8)

Vydala Alfa, vydavateľstvo technickej a ekonomickej literatúry, n. p., 815 89 Bratislava, Hurbanovo nám. 3, ako spoločné vydanie s SNTL — Nakladatelství technické literatury, n. p., Praha, Spálená 51, v novembri 1987 ako svoju 10268. publikáciu

Zodpovedná redaktorka RNDr. Jana Belasová
Jazyková redaktorka PhDr. Katarína Korytárová
Technický redaktor Viktor Slezák
Ochranný obal a väzbu navrhol akad. maliar Peter Galvánek

Vytlačili Západoslovenské tlačiarne, n. p., závod Svornosť, Bratislava
192 strán, 36 obrázkov, 24 tabuliek; 11,95 AH; 12,22 VH
1. vydanie. Náklad 5 000 výtlačkov
302 03 1

063 – 557-87 GAJ Kčs 16,50

104/23; 852